N° d'ordre : 3251

THÈSE

Présentée devant

devant l'Université de Rennes 1

pour obtenir

le grade de : Docteur de l'Université de Rennes 1
Mention Informatique

par

Alban Grastien

Équipe d'accueil : Équipe DREAM - IRISA
École Doctorale : Matisse
Composante universitaire : IFSIC

Titre de la thèse :

Diagnostic décentralisé et en-ligne
de systèmes à événements discrets reconfigurables

soutenue le 13 décembre 2005 devant la commission d'examen

Rapporteurs :			
M^r	Philippe	Dague	(Pr. Université Paris-Sud, président du jury)
M^r	Michel	Combacau	(Pr. Université Paul Sabatier)
Examinateurs :			
M^r	Christophe	Dousson	(CR France Télécom R&D)
M^r	Thierry	Jéron	(CR Irisa / INRIA Rennes)
M^me	Marie-Odile	Cordier	(Pr. Université Rennes 1, directrice de thèse)
M^me	Christine	Largouët	(MdC Agrocampus de Rennes, encadrante)

« Je déteste les citations. »
Ralph Waldo Emerson

Remerciements

Je tiens à remercier M[rs] Philippe Dague et Michel Combacau pour avoir accepté de rapporter cette thèse. Je les remercie d'autant plus que je sais combien leurs emplois du temps étaient chargés en cette fin d'année 2005. Je les remercie également pour les précieuses remarques qu'ils m'ont fournies sur mon manuscript et ma présentation.

Je souhaite également remercier M[rs] Thierry Jéron et Christophe Dousson d'avoir accepté de se joindre au jury de ma soutenance et d'avoir porté autant d'attention à mon travail.

Merci beaucoup à Marie-Odile Cordier et Christine Largouët pour m'avoir permis d'effectuer tout d'abord mon stage de DEA et ensuite cette thèse. Merci à elles de m'avoir encadré aussi efficacement et de m'avoir conseillé durant ces trois années. J'ai le sentiment d'avoir beaucoup appris à leur contact, que ce soit pour la rigueur, la culture et l'écriture scientifiques.

D'une manière plus générale, je voudrais remercier l'équipe DREAM de m'avoir accueilli[1] et en particulier mes deux compagnons de bureau, Élisa[2] et François[3]. Merci à eux de m'avoir tellement fait marrer, qui en chantant *Sex over the phone*, qui en citant *Qui veut la peau de Roger Rabbit ?* ou *Les Monty Pythons*, qui en chantant *Batman*, qui en pétant un cable. Ces trois années à partager votre bureau ont été bien courtes, et vous allez me manquer.

Merci également à tous les gens de l'Irisa avec qui j'ai commencé cette thèse, que ce soit Alexandre[4] (malgré sa cuisine mal adaptée à mon régime alimentaire), Guillaume[5] (malgré ses *Zongo* intempestifs), Gilles[6] (malgré ses blagues), Sabine[7] (malgré son trombone), Gurvan, Stéphane et les autres que j'oublie scandaleusement. Je ne pense pas qu'aux Irisaiens mais également à Jean-Marc[8] qui m'a fait découvrir un pan (voire plusieurs) de la culture musicale et Louis-Marie[9].

Je n'aime pas faire des remerciements parce que je voudrais insérer la moitié de l'Humanité, alors je fais une liste non exhaustive de tous les gens qui m'ont permis de soumettre cette thèse : mes parents, Sonia, Alex Toto, Marion et François-Xavier[10],

[1] Véro, je n'oublie pas que tu as programmé une réunion en 2007.

[2] Apprentissage multisource par programmation logique inductive : application à la caractérisation d'arythmies cardiaques, thèse, décembre 2005

[3] Pilotage d'algorithmes pour la reconnaissance en ligne d'arythmies cardiaques, thèse, décembre 2005

[4] Communications multicast : contributions aux réseaux optiques et au passage à l'échelle, thèse, octobre 2005

[5] Spécification logique de réseaux de Petri, thèse, décembre 2005

[6] Gestion de clés dans les extensions de sécurité DNS, thèse, décembre 2005

[7] Intégration et modélisation de connaissances linguistiques pour la reconnaissance d'écriture manuscrite en-ligne, thèse, décembre 2005

[8] Réduction de la taille des boîtiers MMIC à coup de batte de baseball, thèse, octobre 2005

[9] Je sais pas, j'ai pas pu y aller, thèse, novembre 2005

[10] Vie, mort et résurrection d'un papillon, thèse à six mains, juillet 2005

Onou, Magali[11], Phiip, Jean-Luc Le Ténia[12], M^{r} Havard, M^{me} D., Zongo, François-Xavier Payet qui a travaillé sur RAG$_e$, les Nashville Pussy, Genesis[13]...

[11] Même si ce n'est pas son vrai nom.
[12] Et sa chanson sur la grande Céline (qui fait de l'herpès)
[13] Non, pas Genesis

Table des matières

Introduction

En matière de conception des systèmes complexes, la tendance est d'utiliser des réseaux de composants interreliés aussi appelés *systèmes composites*. On peut trouver de nombreux exemples, depuis les réseaux de télécommunication ou informatiques jusqu'aux systèmes électroniques qui se développent de plus en plus pour les automobiles, en passant par les chaînes de montages dans les usines ou les systèmes robotiques spatiaux. Cette méthode de développement se rapproche de la conception orientée objet utilisée en informatique avec laquelle il existe de nombreux points communs.

Ce schéma de conception, s'il est bien respecté, offre en effet de nombreux avantages. Tout d'abord, la modularité permet de *simplifier* le système. Ainsi, la complexité du système est divisée en une complexité raisonnable pour chaque composant. Par ailleurs, une construction orientée composant offre l'avantage de la *réutilisabilité*, que ce soit sur le même système (plusieurs composants identiques) ou sur différents systèmes (par exemple pour un nouveau modèle de véhicule, il n'est pas nécessaire de reconcevoir à nouveau entièrement les composants mais simplement d'apporter quelques modifications à partir de ceux de la gamme). Enfin, et c'est un des points centraux de cette thèse, la conception modulaire doit permettre une évolution du système par *reconfiguration*.

Une reconfiguration d'un système composite est une modification du système par ajout, suppression ou remplacement de composants, ou par modification de la topologie (connexions entre composants). Une reconfiguration est utile dans de nombreux contextes. Elle permet tout d'abord une mise-à-jour du système par remplacement de composants (obsolètes, dangereux, défectueux, coûteux, etc.) ou par ajout de nouvelles fonctionnalités. En cas de défaillance d'une partie d'un système, il est parfois nécessaire d'isoler cette partie pour éviter une propagation de ce comportement défectueux au sein du système. Dans les réseaux fournissant des ressources, comme par exemple les réseaux de distribution d'électricité, les besoins évoluent de manière régulière. La topologie doit alors s'adapter pour fournir le meilleur service possible.

Ces systèmes, qu'il s'agisse de chaînes de montage, de réseaux de télécommunication, de systèmes embarqués, sont sujets aux pannes et nécessitent d'être diagnostiqués pour comprendre leurs dysfonctionnements et pour réagir à ces mauvais comportements. La surveillance humaine n'est pas toujours possible : risques d'erreurs trop élevés, absence d'humain compétent sur place (par exemple dans le cas des robots spatiaux, ou du suivi d'automobiles), complexité croissante du calcul ou coût trop élevé.

Le *diagnostic* est la partie de l'*intelligence artificielle* visant à détecter, isoler et

identifier les dysfonctionnements d'un système. Le diagnostic s'appuye sur des observations (symptômes) fournies par le système. Ces observations sont parfois imprécises, incertaines, voire partiellement fausses. L'objectif du diagnostic consiste alors à inférer les pannes possibles ou probables sur le système étant données ces observations.

De nombreuses approches ont été développées pour le diagnostic, dépendantes du contexte et des hypothèses sur le système à diagnostiquer. Les méthodes de diagnostic reposant sur un modèle de fonctionnement du système ont longtemps considéré un modèle global du système. La taille du modèle et la complexité du raisonnement nécessaire devenant de plus en plus difficiles à gérer lorsqu'on considère des systèmes de grande taille, de nouvelles techniques, dites de diagnostic décentralisé, sont apparues. Nous nous basons sur ces travaux pour permettre un diagnostic que ce soit *hors-ligne* ou *en-ligne*. Nous nous proposons dans cette thèse d'étendre ces travaux pour prendre en compte de manière plus générale que dans les approches actuelles les incertitudes sur les observations. Nous proposons également de diagnostiquer les systèmes reconfigurables.

Pour cela, nous proposons de représenter les observations sous la forme d'un automate dont chaque trajectoire est une séquence possible d'observations. Nous définissons une structure appelée *chaîne d'automates* permettant de représenter cet automate *par morceaux* et d'effectuer un raisonnement incrémental sur les différents morceaux, que ce soit dans un contexte en-ligne ou hors-ligne. Nous introduisons une seconde structure appelée *chaîne d'automates hétérogènes* permettant de représenter l'évolution du modèle du système lors de l'occurrence de reconfigurations.

Ce travail a été motivé par de nombreux projets auxquels l'équipe DREAM s'est associée, à savoir le diagnostic de réseaux de télécommunication (projets Gaspar, Magda, Magda2), les réseaux électriques, ou des projets annexes pour lesquels l'équipe montre ou a montré un certain intérêt : environnement, *web-services*, systèmes embarqués, robotique spatiale.

Cette thèse est divisée en sept chapitres. Le premier chapitre décrit la problématique en montrant les solutions proposées par la littérature. Ce chapitre est l'occasion de présenter les notions de *système à événements discrets*, *diagnostic*, *observations incertaines*, *calcul incrémental*, *calcul en-ligne*, *diagnostic décentralisé* et *système reconfigurable*. Le deuxième chapitre montre notre approche pour le diagnostic *hors-ligne* avec observations incertaines qui s'appuye sur un modèle et une représentation des observations sous forme d'automates. Le troisième chapitre introduit la notion de chaîne d'automates et montre son utilisation dans le cadre du calcul incrémental du diagnostic. L'automate des observations présenté au chapitre précédent est *découpé* en une chaîne d'automates des observations, et il est possible d'effectuer un raisonnement sur chacun des morceaux de la chaîne pour produire une *chaîne de diagnostics*. Nous proposons plusieurs possibilités pour calculer la chaîne de diagnostics en donnant les intérêts de chacune d'entre elles. Le quatrième chapitre étend les résultats du précédent chapitre au calcul en-ligne du diagnostic. Nous montrons les difficultés qui apparaissent lorsqu'on cherche à construire la chaîne d'automates des observations alors que l'automate des observations n'existe

pas. Le cinquième chapitre s'intéresse au calcul décentralisé et au calcul du diagnostic décentralisé lors du calcul du diagnostic par chaînes d'automates. Les chaînes d'automates permettent en effet de réduire la complexité du calcul, mais les algorithmes doivent être adaptés. Le sixième chapitre étend la définition des chaînes d'automates pour les chaînes d'automates hétérogènes, et montre comment ce formalisme permet de diagnostiquer des systèmes reconfigurables. Le septième et dernier chapitre montre comment nous avons mis en œuvre ces résultats dans l'application appelée RAG_e que nous avons développée.

Chapitre 1

Définition du sujet et approches existantes

Dans ce chapitre, nous présentons les objectifs de cette thèse en définissant les mots-clé de son intitulé. Nous présentons également les méthodes qui nous ont inspirés ou que nous considérons comme importantes pour ces différents mots-clé.

Nous donnons dans un premier temps une présentation informelle de l'objectif de la thèse, puis les mots-clé sont définis de manière plus formelle dans les différentes sections. En section 1.2, nous présentons le *diagnostic de systèmes à événements discrets*. Nous définissons ainsi ce que sont les systèmes à événements discrets et en quoi consiste le diagnostic de ces systèmes. Nous discutons du problème des observations incertaines, point qui est discuté au fil des exemples du chapitre entier. La problématique du diagnostic *en-ligne* est montrée en section 1.3. Dans la section 1.4, nous présentons les méthodes *décentralisées et distribuées*. Finalement, la section 1.5 considère les systèmes à événements discrets *reconfigurables* et les approches développées jusqu'à présent.

La section 1.6 donne un résumé des caractéristiques des algorithmes présentés et explique l'approche que nous avons développée dans cette thèse.

1.1 Objectifs

Cette thèse s'intéresse aux systèmes constitués d'un nombre important de composants interreliés. Nous nous limitons aux systèmes dits *à événements discrets*, c'est-à-dire évoluant de manière discrète à l'occurrence d'événements externes. Sans se restreindre à aucun exemple en particulier, on peut cependant s'appuyer sur l'exemple des réseaux de télécommunication.

Cette thèse se place dans le cadre du diagnostic automatique des systèmes, c'est-à-dire la détection à partir d'*observations* de tout dysfonctionnement (ou *panne*) dans le comportement du système, et l'identification et la localisation cette panne. Le diagnostic s'appuye sur un modèle comportemental (*model-based diagnosis*).

L'objectif de cette thèse est de permettre un diagnostic dans un contexte d'observabilité incertaine. Dans ce contexte, on dispose de toutes les observations sur le système,

mais ces observations ne sont pas sûres. Le raisonnement est donc plus complexe.

L'approche que nous voulons développer doit pouvoir s'appliquer à la fois à un diagnostic *hors-ligne* et à un diagnostic *en-ligne*, c'est-à-dire s'effectuer *a posteriori* de l'exécution du système ou pendant que celui-ci fonctionne.

D'autre part, il est difficile de modéliser les systèmes de grande taille, et encore plus de raisonner sur ce modèle. Aussi, des techniques *décentralisées* et *distribuées* ont été développées, consistant à effectuer un raisonnement de manière locale aux composants avant si nécessaire de passer à un raisonnement global sur le système. Dans le cadre de cette thèse, nous considérons que le diagnostic est effectué par un *superviseur* unique centralisé, et nous ne souhaitons donc pas effectuer un diagnostic distribué. En revanche, notre approche doit pouvoir s'exécuter de manière décentralisée.

Notons enfin que les systèmes que nous considérons peuvent évoluer au cours du temps. Ainsi on peut leur adjoindre de nouveaux composants, modifier les connexions entre les composants, *etc.* Remarquons d'ailleurs qu'une modélisation décentralisée doit permettre une représentation aisée des reconfigurations et de leurs effets. On les qualifie alors de systèmes *reconfigurables*. Le second objectif de cette thèse est de pouvoir diagnostiquer ces systèmes.

Ces points sont détaillés dans les quatre sections qui suivent.

Pour cette thèse, nous nous sommes appuyé sur les travaux de l'équipe DREAM et en particulier les travaux de Y. Pencolé et M.-O. Cordier [PC05]. Cependant, dans ce chapitre, nous présentons de nombreux autres travaux qui nous ont inspiré.

1.2 Diagnostic de systèmes à événements discrets

Dans cette section, nous présentons deux points importants. Tout d'abord, nous définissons les systèmes à événements discrets et nous présentons les différentes manières de modéliser ces systèmes. Ensuite, nous expliquons ce en quoi consiste le diagnostic des systèmes à événements discrets et nous présentons les algorithmes que nous considérons comme représentatifs dans le cadre de cette thèse.

1.2.1 Les systèmes à événements discrets

Cette thèse considère le diagnostic de systèmes à événements discrets. Le TLFi [TLFi] définit un système comme un *ensemble fonctionnel dont les parties sont interconnectées et échangent (...) de la matière, de l'énergie ou de l'information* (définition C.1).

Cassandras et Lafortune (dans [Cas93] puis [CL99]) donnent la définition suivante pour un système à événements discrets : *un système à états discrets et à dynamique événementielle, c'est-à-dire dont l'évolution de l'état dépend entièrement de l'occurrence d'événements discrets et asynchrones au cours du temps.* La dynamique événementielle s'oppose à la dynamique temporelle, où l'état du système évolue spontanément au terme d'un délai. La dynamique temporelle peut cependant être vue de manière événementielle si on représente le terme du délai comme l'occurrence d'un top d'horloge provenant de l'environnement.

Nous choisissons une définition semblable à celle de Cassandras et Lafortune :

Définition 1.1 (Système à événements discrets)**.**
Un système à événements discrets *est un système dynamique pouvant être modélisé par des variables prenant des valeurs dans un domaine discret et évoluant par l'occurrence d'événements discrets et instantanés.*

L'extérieur du système (le *reste du monde*) est appelé l'*environnement* du système.

Un exemple de système à événements discrets est un réseau de télécommunication qui évolue par l'occurrence d'événements extérieurs et par l'envoi de messages d'un composant à un autre. De nombreux systèmes ne sont pas discrets par nature mais peuvent être *discrétisés*. Cette opération consiste à remplacer un domaine de variables continu par un domaine discret. La figure 1.1 donne un exemple de discrétisation d'une fonction. Cette figure montre l'évolution d'une valeur avec le temps dans un domaine continu. Ce domaine est remplacé en un ensemble de 4 valeurs discrètes $\{y_1, y_2, y_3, y_4\}$.

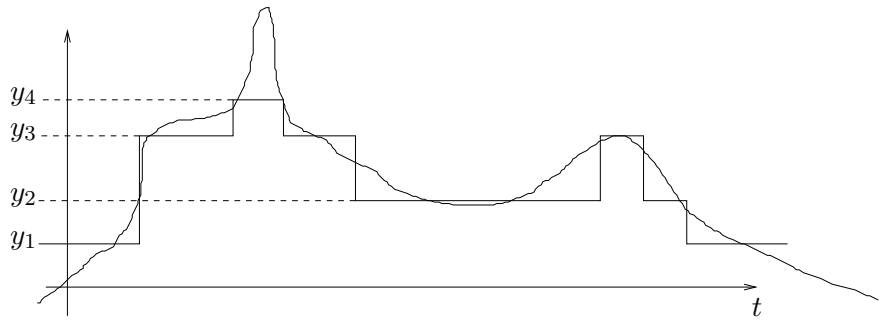

FIG. 1.1 – Exemple de discrétisation

La notion d'événement est très intuitive et difficile à définir. Un événement est une manifestation de l'évolution de l'état du système. Remarquons cependant que l'occurrence d'un événement peut laisser le système inchangé. Un événement est instantané ; un événement n'a pas de durée. Généralement, un événement a une *origine* (l'entité qui produit l'événement) et une *cible* (l'entité qui subit l'événement). Par exemple, si un opérateur appuye sur un bouton d'un composant, alors l'événement est le fait d'appuyer sur le bouton, l'origine est l'opérateur et la cible est le composant.

Lorsque l'événement provient de l'extérieur du système (comme c'est le cas pour l'exemple de l'opérateur), on dit qu'il provient de l'*environnement* et on le qualifie d'*exogène*. Au contraire, si l'origine est une entité du système, alors on dit qu'il s'agit d'un événement *endogène*.

L'occurrence d'un événement peut causer par propagation l'occurrence d'autres événements. Par exemple, si un opérateur appuie sur un interrupteur commandant une ampoule (premier événement), l'ampoule va (normalement) s'allumer (second événement).

Dans les systèmes que nous considérons, un événement endogène est toujours une conséquence d'un événement exogène directement ou indirectement par une séquence d'autres événements endogènes.

Les systèmes les plus étudiés sont les systèmes *réactifs*. Dans ces systèmes, deux événements exogènes ne peuvent avoir lieu en même temps. En effet, le temps est continu et il existe toujours un délai même très faible entre ces deux événements. En revanche, les événements endogènes sont des effets synchrones des événements exogènes, et se produisent donc en même temps que ceux-ci.

Les travaux de G. Lamperti et M. Zanella (décrits en [LZ03b]) considèrent des systèmes *actifs*, c'est-à-dire constitués de composants interreliés qui communiquent entre eux par messages. Dans ces systèmes, certains événements exogènes ciblant un composant conduisent à l'émission synchrone de messages par ce composant vers d'autres composants. L'émission du message a lieu en même temps que l'événement exogène, mais la réception est asynchrone. La réception d'un tel message est ainsi un effet asynchrone de l'événement exogène. L'événement que représente la réception du message peut à nouveau conduire à l'émission de messages vers d'autres composants. Dans les systèmes actifs comme dans les systèmes réactifs, un seul événement déclencheur (événement exogène ou réception asynchrone de message) a lieu à un instant donné.

1.2.2 Différentes modélisations des systèmes à événements discrets

Le TLFi [TLFi] définit le modèle comme un *système physique, mathématique ou logique représentant les structures essentielles d'une réalité et capable à son niveau d'en expliquer ou d'en reproduire dynamiquement le fonctionnement* (définition C.1). Nous nous inspirons de cette définition pour donner notre propre définition dans notre contexte :

Définition 1.2 (Modèle)**.**
Le modèle *est une représentation mathématique permettant de simuler le comportement d'un système.*

Le modèle d'un système est par définition une abstraction du comportement réel du système. Cette abstraction peut se faire à différents niveaux. Par exemple, pour un circuit électrique, on peut choisir de modéliser le comportement de chaque électron, ou abstraire ce comportement et ne garder que les relations entre la tension, l'intensité et la résistance. L'abstraction d'un modèle permet un calcul plus rapide (car un modèle abstrait est un modèle simplifié) et un résultat généralement plus facile à comprendre pour un opérateur humain (voir [TT03b]). Le modèle peut aussi ne capturer qu'une partie des comportements, considérant que certains comportements sont mal connus ou ne nécessitent pas d'être modélisés. C'est ainsi qu'on peut avoir le *modèle de bon comportement* (c'est-à-dire le modèle décrivant le fonctionnement correct du système) et ne pas avoir le *modèle de mauvais comportement*.

La modélisation est choisie en fonction du système et de la tâche que l'on souhaite effectuer à l'aide de ce modèle. Ainsi, les modèles pour le diagnostic ne sont pas nécessairement les mêmes que les modèles utilisés pour la planification. Nous donnons ici

quelques modélisations utilisées dans le cadre du diagnostic général. Notons qu'il est possible de *compiler* le modèle [HD05] pour obtenir un modèle adapté à la tâche pour laquelle il est créé.

1.2.2.1 Algèbres de processus

Les algèbres de processus s'appuyent sur la théorie des langages pour représenter le comportement des systèmes. Dans ce formalisme, un *mot* du langage est une séquence possible d'événements dans le système.

On peut notamment citer PEPA [HG], [Hil94], [GH94]. PEPA est une algèbre de processus stochastique permettant de modéliser des comportements. PEPA permet d'étudier les propriétés comportementales ou de performance du système. PEPA est utilisé pour de nombreuses applications (diagnostic par systèmes experts, protocoles pour systèmes tolérants aux fautes, réseaux de téléphones cellulaires...).

Dans PEPA, un événement est appelé *activité* et noté $a = (\alpha, r)$. L'*étiquette* ou *type d'action* $\alpha \in \mathcal{A}$ identifie l'action, et le *taux d'activité* r est le paramètre exprimant sa durée. Le paramètre r n'est pas utilisé dans le cadre du diagnostic, et nous le laissons donc de côté. PEPA utilise un petit groupe d'opérateurs pour spécifier le langage.

$$\begin{array}{lll} S & ::= & A \mid \alpha.S_1 \mid S_1 + S_2 \qquad \text{(comportement du composant)} \\ P & ::= & S \mid P_1 \underset{L}{\bowtie} P_2 \mid P_1/H \qquad \text{(structure du système)} \end{array}$$

Constante : L'équation $S \stackrel{def}{=} A$ donne à S le comportement du composant A. Par ce biais, il est possible de définir des comportements récursifs.

Préfixe : C'est le mécanisme de base de comportement d'un composant. $\alpha.S_1$ effectue l'action α avant de se comporter comme S_1.

Choix : L'opérateur de choix indique la possibilité pour le composant de se comporter de différentes manières. Ainsi, dans la formule, $S ::= S_1 + S_2$, le comportement sera celui de S_1 ou celui de S_2.

Coopération : La coopération permet d'effectuer le produit de plusieurs modules comme nous le voyons par la suite pour la modélisation décentralisée ; la coopération est également utilisée dans le cadre du diagnostic. $P ::= P_1 \underset{L}{\bowtie} P_2$ signifie que les mots de P_1 et P_2 sont synchronisés sur les lettres de L.

Masquage : L'opération P_1/H permet de masquer les actions de H qui sont ainsi considérées comme internes.

On peut reprendre l'exemple du modèle de la pompe fourni dans [CPR00]. Dans cet exemple, une pompe fournit de l'eau. Elle peut être en bon état, en fuite ou en panne.

$$\begin{array}{lll} P & \stackrel{def}{=} & Pok_1 + Plk_1 + Pbl_1 + End \\ Pok_1 & \stackrel{def}{=} & nrm_0.nrm_P.P \\ Plk_1 & \stackrel{def}{=} & nrm_0.low_P.P \\ Pbl_1 & \stackrel{def}{=} & nrm_0.zro_P.P \end{array}$$

Dans cet exemple, la pompe peut fonctionner de trois manières : Pok_1, Plk_1 ou Pbl_1 (*End* représente la fin de la période considérée). Lorsque la pompe est en mode Pok_1, l'occurrence de nrm_0 (entrée d'eau normale) conduit à l'occurrence de nrm_P (sortie d'eau normale). Après quoi, le terminal P indique que le système peut à nouveau changer de mode de comportement.

On trouve d'autres formalismes se basant sur la théorie des langages comme dans [Sen98], [SW04].

1.2.2.2 Réseaux de Petri

Les *réseaux de Petri* [Mur89] sont un outil particulièrement adapté à la représentation des ressources dans un système.

Formellement, un réseau de Petri est un 5-tuple $PN = (P, T, F, W, M_0)$ où :
- P est un ensemble de places,
- T est un ensemble de transitions avec $P \cap T = \emptyset$ et $P \cup T \neq \emptyset$,
- $F \subseteq (P \times T) \cup (T \times P)$ est l'ensemble des arcs,
- $W : F \times \mathbf{N}^+$ est une fonction de poids et
- $M_0 : P \rightarrow \mathbf{N}$ est le marquage initial,

Généralement, une place représente une ressource, et une transition représente une action ou un événement. Les arcs de F indiquent les ressources consommées ou produites par un événement, tandis que W indique la quantité de ces ressouces.

Le *marquage* d'un réseau de Petri est une fonction $M : P \rightarrow \mathbf{N}$. Le marquage indique le nombre d'instances (appelées *jetons*) de chaque ressource de P. En particulier, M_0 indique le nombre de ressources dont dispose le système au début de l'exécution. La dynamique des réseaux de Petri est régie par les transitions. Une transition t est *tirable* si chaque place p reliée à la transition par un arc $(p, t) \in F$ dispose d'un nombre de jetons supérieur ou égal au poids de cet arc $W(p, t)$. Dans le cas positif, et si la transition est tirée, chaque place p reliée par un arc $(p, t) \in F$ à la transition dépense un nombre de jetons égal au poids de cet arc $W(p, t)$, et chaque place p reliée à la transition t par un arc $(t, p) \in F$ reçoit un nombre de jetons égal au poids de cet arc $W(t, p)$. Dans le cas contraire, la transition n'est pas tirable.

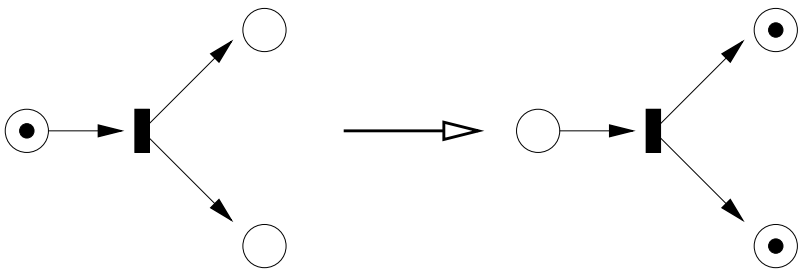

FIG. 1.2 – Illustration du franchissement d'une transition

La figure 1.2 donne un exemple de franchissement de transition. Les places sont représentées par des cercles et les transitions par des rectangles. Les arcs sont représentés

par des flèches reliant le premier argument au second. Le poids est indiqué à côté de la transition si celui-ci est supérieur ou égal à deux. Il vaut 1 dans le cas contraire. Avant le tir, la place de gauche dispose d'un jeton (représenté par le disque noir). La transition est donc tirable. Le franchissement de la transition consomme le jeton de gauche et produit un jeton dans chacune des deux places à droite de la transition. On voit qu'il est facile de représenter de cette manière des processus utilisant des ressources. Ainsi, effectuer l'action de la figure 1.2 coûte une ressource du type de la place de gauche et produit deux ressources, une pour chaque type des places de droite. Notons qu'une transition peut ne pas comporter de flèche entrante (événement spontané qui ne consomme pas de ressource), ou au contraire ne comporter aucune flèche sortante (événement consommant une ressource sans en produire).

On peut noter que dans certaines notations, le nombre de jetons associé à une place donnée prend des valeurs dans un espace borné $\{0, \ldots, n\}$ voire binaire $\{0, 1\}$ (réseaux de Petri sûrs, *safe Petri nets*). Pour ces modélisations, il est ainsi nécessaire de s'assurer que le nombre de jetons de chaque place restera borné quelle que soit l'évolution du système. La figure 1.3 donne un exemple de réseau de Petri complet. Dans cet exemple, le marquage initial comprend trois jetons, dont deux pour la place P_0 et un pour la place P_8. Les différentes couleurs des transitions ont une sémantique pour le diagnostic, et sont donc expliquées plus loin.

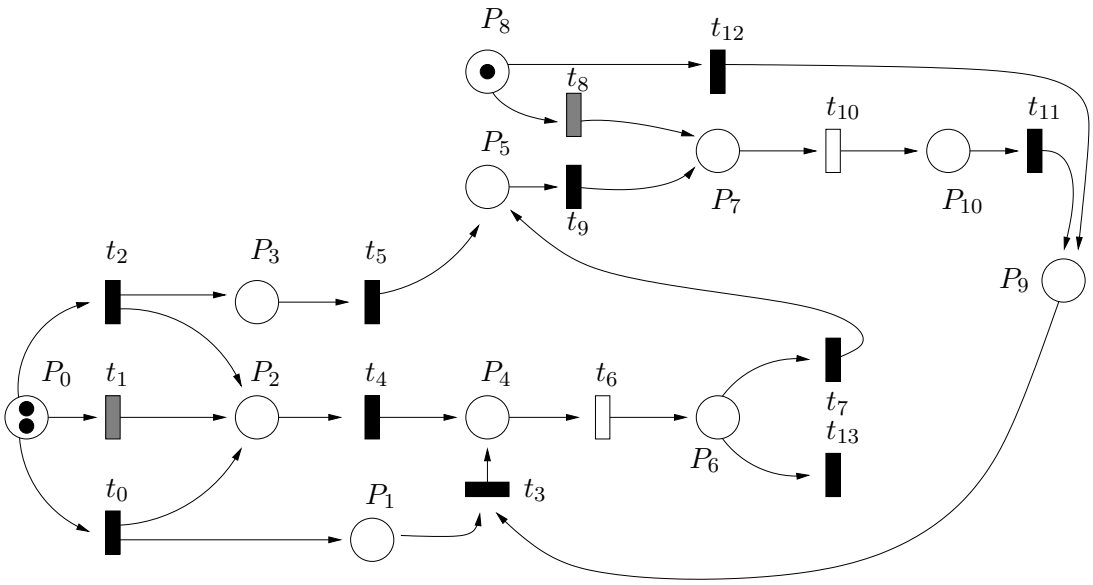

FIG. 1.3 – Exemple de réseau de Petri

Plusieurs approches ont été développées pour introduire un aspect probabiliste dans les réseaux de Petri : réseaux de Petri stochastiques [DA94], réseaux de Petri stochastiques généralisés [MBC+95], réseaux de Petri partiellement stochastiques [AFB+98].

L'intérêt des réseaux de Petri utilisant des probabilités est de ne conserver que les comportements les plus probables.

1.2.2.3 Automates

Un automate est un ensemble d'états et de transitions menant d'un état à un autre état. Les transitions sont étiquetées par les événements menant d'un état à un autre état.

Il existe de nombreuses définitions équivalentes d'automates. Nous avons choisi la définition $A = (Q, E, T, I, F)$ présentée en annexe A. Dans cette définition, Q est l'ensemble des états possibles du système et E est l'ensemble des événements qui peuvent avoir lieu sur le système. T est l'ensemble des transitions $t = (q, l, q')$ où $q \in Q$, $q' \in Q$ et $l \subseteq E$. La transition t indique que l'occurrence simultanée des événements de l lorsque le système est dans l'état q le conduit à l'état q'. L'ensemble des états initiaux I est l'ensemble des états possibles au début de la période que l'on étudie. L'ensemble des états finaux F est l'ensemble des états possibles à l'issue de la période que l'on étudie.

Remarquons qu'il existe quelques différences d'une notation à une autre. Ainsi, certaines notations ignorent l'ensemble des états finaux F (tous les états sont finaux). Si on considère le cas déterministe, alors I ne comprend qu'un seul état et T est une fonction partielle déterministe qui à un état q et un ensemble d'événements l associe un unique état q' ($T : (Q \times 2^E) \rightarrow Q$). Enfin, certaines notations considèrent qu'un seul événement a lieu à une date donnée, et donc $T \subseteq Q \times E \times Q$. La figure 1.4 donne un exemple d'automate.

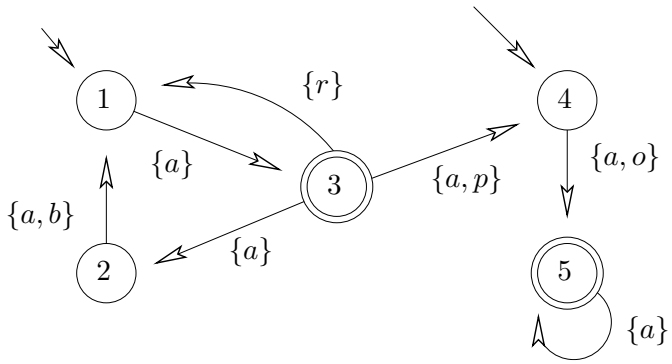

FIG. 1.4 – Exemple d'automate modélisant un comportement

Un problème associé aux automates est la représentation des comportements concurrents. Deux ensembles d'événements l_1 et l_2 sont concurrents lorsqu'ils interviennent sur deux parties disjointes du système. Dès lors, lorsque les deux comportements sont possibles, il peuvent avoir lieu dans n'importe quel ordre. La figure 1.5 montre ce qui se passe avec les deux ensembles l_1 et l_2.

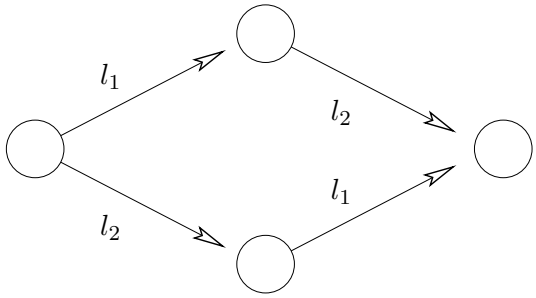

FIG. 1.5 – Exemple du problème de concurrence dans un automate

Lorsqu'on considère de gros systèmes, de très nombreux comportements sont concurrents. Le nombre d'états nécessaires pour réprésenter n ensembles d'événements concurrents et le nombre des transitions sont exponentiels. Il est cependant possible de contourner ce problème par l'utilisation de techniques de réduction d'ordre partiel [Pel93] utilisant les traces de Mazurkiewicz [Maz88]. Une trace de Mazurkiewicz est une séquence d'événements qui représente l'ensemble des séquences d'événements qui peuvent être obtenues en intervertissant deux événements successifs concurrents. Ainsi, on peut représenter un automate de manière *réduite*. Ces techniques sont cependant difficiles à mettre en place et un automate réduit est difficile à calculer et manipuler.

Remarquons que le problème de concurrence n'apparait pas dans les réseaux de Petri. Ainsi, dans l'exemple de la figure 1.3, on voit que les transitions t_2 et t_{12} sont indépendantes. D'autre part, ces techniques ne parviennent que partiellement à réduire la taille des automates.

Il existe également des modélisations incluant des notions de temps. Il est alors possible de modéliser la distribution de probabilité du délai d'occurrence entre certains événements. On peut ainsi citer les automates temporisés [Alu99].

Un automate temporisé est un tuple $\mathcal{A} = (\mathcal{S}, \mathcal{X}, \mathcal{L}, \mathcal{E}, \mathcal{I})$ où $\mathcal{S}$ est un ensemble fini de locations, $\mathcal{X}$ est un ensemble fini d'horloges et $\mathcal{L}$ est un ensemble fini d'étiquettes. Un état du système est modélisé par un couple (s, v) comprenant une location et une valuation de chaque horloge (dans $\mathbf{R}^+$). $\mathcal{I}$ donne une contrainte entre la location et la valuation des horloges. Une transition $e = (s, l, \phi, \delta, s')$ connecte la location s à la location s' et est étiquetée par $l \in \mathcal{L}$. La transition est possible si la valuation des horloges respecte la formule ϕ (conjonction de comparaisons entre la valuation d'une horloge et une constante) et entraîne une remise à zéro des horloges de $\delta \subseteq \mathcal{X}$. La formule ϕ est appelée la *garde* de la transition.

La sémantique des automates temporisés est définie par le système de transitions suivant : $(\mathcal{Q}, \rightarrow, (s_0, v_0))$ où $\mathcal{Q}$ est l'ensemble (infini) des états, $\rightarrow$ est l'ensemble (infini) des transitions et (s_0, v_0) est l'état initial ($s_0 \in \mathcal{S}$ et $\forall x \in \mathcal{X}$, $v_0(x) = 0$). Il existe deux types de transitions.

– Transitions discrètes correspondant à l'occurrence d'un événement. Soit $e =$

$(s, l, \phi, \delta, s') \in \mathcal{E}$. Alors, il existe une transition étiquetée par l entre (s, v) et (s', v') si v satisfait ϕ et v' est la valuation v où toutes les horloges de δ ont été remises à zéro.

– Transitions temporisées correspondant à l'écoulement du temps. Soit $t \in \mathbf{R}^+$. Alors, il existe une transition entre (s, v) et (s, v_t) avec v_t la valuation obtenue en ajoutant t à la valuation de chaque horloge si pour tout $t' < t$, $v_{t'}$ satisfait $\mathcal{I}(s)$.

Deux transitions discrètes ne peuvent avoir lieu successivement ; une transition temporisée doit s'insérer entre ces deux transitions.

On peut citer les outils KRONOS [Yov97] et UPPAAL [BY04, Dav] qui peuvent être utilisés pour vérifier des propriétés sur des systèmes modélisés par automates temporisés. Notons qu'UPPAAL repose sur le formalisme des automates temporisés mais comprend également des transitions qualifiées d'*urgentes* qui doivent être franchies immédiatement lorsqu'elles deviennent franchissables.

Comme nous l'avons vu, le nombre d'état d'un automate temporisé est infini. Le raisonnement sur de tels automates est donc complexe [BBF+01a]. Il est cependant possible de réduire en partie cette complexité par des structures de données appelées *difference-bound matrices*.

Des modélisations existent également pour prendre en compte la probabilité d'occurrence d'un événement plutôt qu'un autre. Le modèle attache alors à chaque transition une probabilité. La probabilité d'un chemin est alors le produit des probabilités de chaque transition empruntée. Ces probabilités permettent d'estimer la plus grande probabilité d'évolution en cas d'incertitude sur l'évolution possible (voir le chapitre 6 de [Lar00] et [TT03a]).

La taille des modèles sous forme d'automates dans de nombreux cas réels est très importante. Il est donc nécessaire d'utiliser différentes techniques pour résoudre ce problème. C'est le cas des approches décentralisées présentées par la suite en section 1.4. Une autre technique est de représenter l'automate de manière symbolique par exemple par $p-$DD ou BDD (voir par exemple [SM96, MR02, SPT04]). Les $p-$DD sont une généralisation des BDD. Dans un BDD ((*ordered*) *Binary Decision Diagram*), chaque état du système est représenté par une conjonction de propriétés. Les propriétés permettent de représenter complètement l'état pour le niveau d'abstraction choisi ; deux états ne sont donc pas modélisés par la même conjonction de propriétés. Un ensemble d'états est alors représenté par une disjonction de formules, où chaque formule est la représentation d'un des états de l'ensemble. L'ensemble d'états est donc représenté par une disjonction de conjonctions. Par factorisation, il est parfois possible de représenter un ensemble de manière compacte. Une transition dans un automate est représentée comme la conjonction des représentations de l'état source, de l'ensemble des événements et de l'état cible. À nouveau, l'ensemble des transitions est représenté par la disjonction des formules représentant les transitions.

Les BDD ont de nombreux intérêts :

– la représentation est souvent compacte (notamment si on choisit un bon ordonnancement pour les variables [TT04]),

- la représentation est canonique pour un ordonnancement des variables donné,
- les opérations sont simples à implémenter et efficaces (complexités constante, linéaire ou quadratique).

Dans le cadre du diagnostic par automates, H. Marchand et L. Rozé [MR02] proposent une *réduction symbolique* du modèle. Intuitivement, deux états x_1 et x_2 sont dits *équivalents*, noté $x_1 \sim x_2$, si l'occurrence d'un événement y depuis x_1 mène à un état x'_1 équivalent (ou égal) à x'_2 atteint par l'occurrence de y en x_2, et *vice versa*. Deux états équivalents $x_1 \sim x_2$ sont impossibles à différencier grâce aux observations. Il est alors possible de construire un *modèle quotient* dont les états sont remplacés par des classes d'états équivalents. L'automate produit est généralement beaucoup plus petit, et plus facile à manipuler.

1.2.2.4 Choix de la modélisation

Les correspondances entre les différents types de modélisations sont très fortes. Ainsi, il est possible de générer un automate à partir d'un réseau de Petri : c'est ce qu'on appelle le *graphe de cas* (*case graph*). Remarquons cependant que si le nombre de jetons dans une place n'est pas borné, alors l'automate n'est pas de taille finie. Par ailleurs, une modélisation d'automates par BDD est très semblable à une modélisation par réseau de Petri sûrs. Dans ce cas, chaque variable d'état du BDD correspond à une place du réseau de Petri.

D'autre part, les définitions utilisées dans le cadre du diagnostic par automate raisonnent généralement sur les langages reconnus par l'automate (voir par exemple [KGM91]). La relation avec les algèbres de processus est donc immédiate.

Nous avons repris pour cette thèse le formalisme utilisé au sein de l'équipe lors des précédents travaux. En particulier, nous utilisons une modélisation basée sur l'utilisation d'automates. Nous avons vu que la principale difficulté est la taille du modèle. Ce point est résolu par l'utilisation d'une modélisation décentralisée présentée dans la section 1.4.

Nous considérons que les problématiques associées à l'utilisation d'un automate sous forme de BDD sont relativement indépendantes du problème qui nous intéresse dans cette thèse. D'autre part, l'intérêt des BDD dans un contexte décentralisé n'est pas important, puisque le nombre d'états est faible. Aussi, nous n'avons pas utilisé une modélisation par BDD mais par énumération des états.

1.2.3 Le diagnostic de systèmes à événements discrets

Le diagnostic est un domaine de l'intelligence artificielle consistant à détecter, localiser et si possible identifier précisément tout dysfonctionnement au sein d'un système. Dans cette sous-section, nous définissons le diagnostic dans le cadre de systèmes à événements discrets. Pour cela, nous définissons les dysfonctionnements, appelés *pannes*. Nous définissons ensuite les informations connues sur le comportement du système, appelées *observations* et expliquons le rôle du superviseur en charge du diagnostic. Puis,

nous définissons le diagnostic. Enfin, nous discutons d'une problématique proche appelée *diagnosticabilité*.

1.2.3.1 Pannes

Il existe différentes définitions de panne. [TLFi] donne la définition suivante : *arrêt momentané accidentel et subit du fonctionnement d'un mécanisme, d'un moteur, d'un appareil ; impossibilité de fonctionner* (définition 3 B). On peut donc considérer que la panne est le fait pour le système d'être en fonctionnement anormal.

Définition 1.3 (Panne)**.**
Une panne *est un ensemble d'états du système correspondant à un dysfonctionnement.*

Dans le cadre du diagnostic, on peut également être intéressé par la cause de la panne, c'est-à-dire l'événement qui a causé la panne.

Définition 1.4 (Événement de panne)**.**
Un événement de panne *est l'événement conduisant le système dans un état de panne.*

Un événement de panne est aussi appelé une *faute*. Dans certains contextes, les pannes sont *intermittentes*. Dans ce cas, il existe des événements permettant au système de recouvrir un état de comportement normal. Dans le cas contraire, la panne est qualifiée de *permanente*.

Si on considère un système comme un ensemble de composants, le fait qu'un composant tombe en panne peut conduire d'autres composants à tomber en panne. Ainsi, si un composant électrique provoque un court-circuit, non seulement il tombe en panne, mais il risque d'endommager tous les autres composants électriques. Ce phénomène est désigné par le terme de *propagation des pannes*. Selon le contexte, une information pertinente pour le diagnostic est l'ensemble des pannes sur le système ou l'événement de panne original.

1.2.3.2 Observations

Le système comporte un ensemble d'événements observables noté E_{Obs}. L'occurrence d'un événement observable sur le système génère une *observation émise*. Une définition générale d'observation émise est difficile à donner pour rester suffisament général. Nous donnons cette définition :

Définition 1.5 (Observation émise)**.**
Une observation émise *est une information sur le comportement interne du système résultant de l'occurrence d'un événement observable.*

Physiquement, certaines observations sont en réalité générées par des capteurs extérieurs au système. Par abus de langage, nous disons que les observations sont émises par le système. Selon le type de capteurs ou de systèmes, une observation peut contenir différentes informations. Généralement, une observation contient l'information de l'événement observable ayant généré l'observation. Dans certains cas, le capteur peut étiqueter l'observation avec la date d'émission de l'observation ou donner un numéro à l'observation.

De manière générale, on ne dispose pas des observations émises par le système. Les observations émises ne sont pas fournies directement mais sont transmises *via* des systèmes électroniques (dont les capteurs) que nous appelons *canaux de communication*. On suppose qu'on dispose de l'ensemble des observations reçues.

Définition 1.6 (Observation reçue)**.**
Une observation reçue *est une information dont on dispose sur une observation émise par le système.*

Remarquons qu'une observation reçue peut être extrêmement différente d'une observation émise. La tâche de reconstruction des observations émises à partir des observations reçues n'est pas nécessairement évidente. Il est donc nécessaire de connaître le comportement des canaux de communication pour retrouver les observations émises. On parle de *complétude* lorsque l'ensemble des observations reçues est le même que l'ensemble des observations émises.

Le cas le plus simple, et celui qui est considéré le plus souvent, est le cas où la séquence des observations émises est identique à la séquence des observations reçues.

Prenons maintenant le cas où toutes les observations émises sont reçues avec un délai, et toute observation reçue correspond à une observation émise. Dans ce cas, le problème est de retrouver l'ordre des observations. Comme nous l'avons indiqué, les observations émises (et donc les observations reçues) peuvent être étiquetées par la date d'émission. Cependant, si les capteurs ne sont pas synchronisés, alors les dates d'émission provenant de différents capteurs ne sont pas comparables. Si le délai de transmission d'une observation est borné, il est possible de donner l'ordre d'émission de certaines observations grâce à la date de réception. On dispose donc d'un ensemble partiellement ordonné d'observations émises.

Dans certains contextes, les canaux de communication peuvent fournir des observations incertaines ou erronées. Lors de la transmission des observations, celles-ci sont transformées. Il est même possible qu'une observation reçue par le superviseur n'ait pas été émise par le système ; il s'agit d'un bruit sur les canaux de communication.

Enfin, on peut trouver des cas où les observations émises ne sont jamais reçues. Il arrive ainsi qu'une observation soit perdue pendant la transmission sur les canaux de communication. Il arrive également que les observations soient en partie transmises par le système, notamment dans le cas des réseaux de télécommunication. Dans ce cas, si un des composants tombe en panne, il ne peut plus transmettre les observations. C'est ce que l'on désigne sous le terme de *masquage des observations*.

On appelle *fenêtre d'observations* un découpage des observations émises par le système, correspondant à un sous-ensemble des observations émises. Si ce découpage est fait entre deux dates, on parle de *fenêtre temporelle*.

Cette section considère le diagnostic *hors-ligne*. Dans ce contexte, toutes les observations devant être reçues le sont effectivement. Le fait d'attendre plus longtemps ne permet pas d'obtenir d'observations supplémentaires. On dispose donc de la fenêtre complète des observations. Les seules observations qui n'ont pas été reçues sont celles qui ont été perdues.

1.2.3.3 Diagnostic hors-ligne

La définition générale du diagnostic est la suivante :

Définition 1.7 (Diagnostic)**.**
Le diagnostic *consiste à détecter toute panne sur un système, l'identifier et la localiser.*

Par défaut et sauf mention contraire, nous considérons le diagnostic hors-ligne. Le cas *en-ligne* est présenté en section 1.3. La tâche du diagnostic s'appuie sur le modèle du système et les observations pour trouver les pannes sur le système.

Le diagnostic est donc un ensemble de pannes (appelé *candidate diagnosis*, voir par exemple [CPR00], [LZ03b]). Il n'est généralement pas possible de trouver un unique diagnostic candidat ; aussi le diagnostic est plus généralement un ensemble d'ensemble de pannes.

Dans le contexte des systèmes à événements discrets, on considère que la tâche de diagnostic revient à retrouver la ou les séquences d'événements qui a eu lieu sur le système, et les états par lesquels celui-ci est passé. Cette séquence peut être appelée *scenario* [CT94], *history* [BLPZ99], *narrative* [BMS00], *consistent path* [CPR00] ou *trajectoire* [CGLP03a]. Nous choisissons ce dernier terme par la suite. À partir d'une trajectoire, il est facile de retrouver quelles pannes ont eu lieu (voir par exemple les algorithmes développés dans [LZ03b]). Nous ne considérons pas cette partie du diagnostic dans cette thèse. Il n'est généralement pas possible de trouver la séquence d'événements sur le système. Le diagnostic consiste alors à chercher l'ensemble des trajectoires possibles.

Définition 1.8 (Diagnostic de système à événements discrets)**.**
Le diagnostic de système à événements discrets *consiste à calculer l'ensemble des trajectoires sur le modèle du système qui peuvent génèrer les observations reçues.*

Dans [PC05], les observations reçues sont partiellement ordonnées. Cela signifie que les observations émises par le système ne sont pas reçues dans l'ordre de leur émission. Le *comportement observé* est alors défini comme l'ensemble partiellement ordonné des observations émises. L'article définit également un *comportement observable* d'une trajectoire. Le comportement observable d'une trajectoire est l'ensemble des observations émises au cours de cette trajectoire, c'est-à-dire la projection de la trajectoire sur l'ensemble des événements observables. Le diagnostic peut alors se définir comme l'ensemble des trajectoires sur le modèle telles que le comportement observable de ces trajectoires est un des ordonnancements du comportement observé.

1.2.3.4 Diagnosticabilité

Une notion supplémentaire a été définie appelée *diagnosticabilité* (*diagnosability* en anglais, parfois simplifié en *diagnosabilité* en français). Un système est dit diagnosticable s'il est toujours possible de déterminer quelle catégorie de panne a eu lieu sur le système, quelle que soit l'évolution du système. Formellement, on trouve cette définition [SSL+95] :

$$(\forall i \in \Pi_f)(\exists n_i \in \mathbf{N})[\forall s \in \Psi(\Sigma_{fi})](\forall t \in L/s)[\|t\| \geq n_i \Rightarrow (\omega \in P_L^{-1}[P(st)] \Rightarrow \Sigma_{fi} \in \omega)]$$

Cette formule complexe indique la notion suivante : pour tout type d'événement de panne f_i, il existe un nombre n_i tel que si on considère une séquence d'événements s comprenant l'événement f_i, alors au terme de n événements t consécutifs à s (avec $n \geq n_i$), il sera possible de diagnostiquer avec certitude que l'événement f_i a eu lieu. Cela signifie qu'il suffit d'attendre un nombre d'événements borné pour s'assurer du bon ou du mauvais comportement d'un système. On considère de plus généralement que le système ne peut pas tomber dans un état où il ne génère plus d'observations. Notamment, on considère qu'il n'y a pas d'*impasse* (*deadlock*), c'est-à-dire que le système ne peut pas être bloqué dans un état et qu'il n'y a pas de boucle sans observations (ou alors, une politique dite d'*impartialité* (*fairness*) indique que le système ne boucle pas indéfiniment sans générer d'observation, même lorsque le modèle le permet).

On peut trouver différentes formulations pour cette notion, notamment dans le cadre décentralisé ou distribué (voir 1.4) ou considérant un temps minimum à attendre t_i plutôt qu'un nombre d'observations pour détecter un événement de panne.

La diagnosticabilité permet de s'assurer que le système est suffisamment observable pour discriminer les différents types de comportement. Si ce n'est pas le cas, il convient d'ajouter des capteurs au système. La problématique de diagnosticabilité est proche du problème de diagnostic. Nous voyons par la suite une utilité supplémentaire dans le cadre du diagnostic décentralisé (voir sous-section 1.4.3.1, page 36). Notons cependant que cette thèse ne s'intéresse pas à la caractérisation de la diagnosticabilité des systèmes.

1.2.4 Les algorithmes de diagnostic

Nous présentons maintenant les techniques de diagnostic développées dans la littérature pour le diagnostic centralisé hors-ligne des systèmes à événements discrets.

1.2.4.1 Algèbres de processus

Nous reprenons le cas de Pepa que nous avons présenté dans 1.2.2.1. Dans ce cadre, le modèle est noté SD (pour *system description*).

Observations : L'article [CPR00] ne donne pas de précision sur la forme des observations reçues, mais considère que les observations émises peuvent être représentées comme un langage décrit par le formalisme de Pepa. Ce formalisme offre une grande expressivité pour représenter des observations incertaines, partiellement ordonnées ou des observations perdues. Par exemple, $Obs \stackrel{def}{=} nrm_p.end.End + low_p.end.End$ indique que l'on a reçu une seule observation nrm_p ou low_p (le terminal end et le non terminal End indiquent la fin des observations).

Le diagnostic est défini comme l'ensemble des diagnostics candidats, où un diagnostic candidat est un ensemble d'événements de panne. Ce résultat peut être obtenu grâce à la formule suivante :

$$Diag \stackrel{def}{=} (Obs \underset{S\cup\{end\}}{\bowtie} SD) \,/H$$

S contient l'ensemble des événements observables du système. La coopération entre Obs et SD sur l'ensemble $S \cup \{end\}$ fournit l'ensemble des mots de SD dont les observations forment un mot de Obs. Dans le contexte, un mot de SD est un comportement du système et un mot de Obs est une séquence possible d'observations émises. Cela signifie qu'on ne conserve que les comportements de SD tels que la projection sur l'ensemble des observations (le comportement observable que nous avons présenté avant) se synchronise avec les observations émises (comportement observé).

Dans la formule du diagnostic, H est l'ensemble des événements qui ne nous intéressent pas dans le cadre du diagnostic. Par exemple, H comprend les événements internes. Le masquage de ces événements permet de ne conserver du comportement que les événements de panne. Ainsi, $Diag$ comprend uniquement les événements de panne qui ont pu se produire dans le système.

1.2.4.2 Réseaux de Petri

Dans le contexte du diagnostic de système modélisé par un réseau de Petri, les transitions sont partionnées en trois catégories : événements non observables, événements observables et événements de panne. Sur la figure 1.3, ces trois types d'événements sont représentés respectivement par des transitions de couleur noire, blanches et grises.

Observations : Le diagnostic par réseaux de Petri considère que les observations sont complètement connues, mais autorise un ordre partiel sur les observations comme nous le voyons par la suite.

Le diagnostic par réseaux de Petri (voir par exemple [BFHJ03]) se base sur l'opération de dépliage [NPW81]. Cette opération permet d'obtenir le marquage d'un réseau à la suite d'une séquence d'événements. La figure 1.6 montre un exemple de dépliage du réseau de Petri de la figure 1.3. Le dépliage d'un réseau de Petri est un homomorphisme du réseau. Pour simplifier, dans un dépliage, plusieurs places peuvent avoir le même nom. Les places qui n'ont pas de prédécesseurs (ici deux places P_0 et une place P_8) correspondent au marquage initial. Les transitions ont les mêmes prédécesseurs que dans le réseau. Enfin, chaque place n'a qu'un seul prédécesseur et il n'y a pas de boucle. Un dépliage est une configuration lorsqu'aucune place n'a plusieurs successeurs, ce qui est le cas sur la figure 1.6. Les places d'une configuration qui n'ont pas de successeurs représentent le marquage final (ici, P_0, P_2 et P_6). Le dépliage montre l'évolution du système lors de l'occurrence des événements représentés par les transitions, ici t_0, t_{12}, t_3 et t_6. Ces événements ont un ordre partiel, puisque t_3 a eu lieu après t_{12} (la transition t_3 succède à t_{12}), mais en revanche l'ordre entre t_0 et t_{12} est inconnu. Le dépliage des réseaux de Petri permet donc de représenter de manière efficace des comportements concurrents où l'ordre des événements concurrents n'est pas important. Notamment, cette représentation est intéressante en cas d'ordre partiel sur les observations.

L'évolution présentée sur la figure 1.6 est l'une des explications de l'observation t_6. Il est possible d'en trouver d'autres. La technique utilisée est le *branching process* qui consiste à représenter de manière compacte un ensemble de dépliages. Dans un

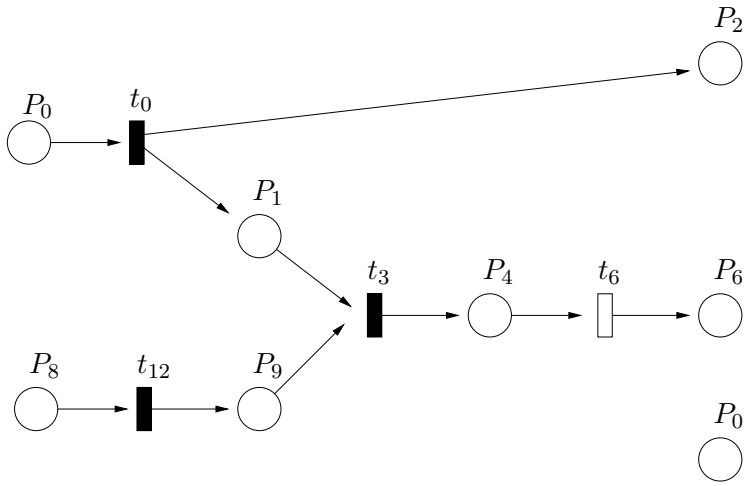

FIG. 1.6 – Dépliage du réseau de Petri de la figure 1.3

branching process, les places peuvent avoir plusieurs transitions successeurs. Une seule de ces transitions a été empruntée dans la réalité. Ainsi, emprunter une transition peut impliquer qu'une ou plusieurs autres transitions ne peuvent plus être empruntées. On dit qu'elles sont en *conflit*.

Une autre méthode proposée utilise un calcul par retour arrière [JB05]. Dans cette méthode, un raisonnement abductif, c'est-à-dire se basant sur les observations, est utilisé pour trouver les causes de ces observations (en particulier, les pannes). L'algorithme utilise un calcul en arrière et obtient pour chaque séquence (partiellement ordonnée) le marquage minimum nécessaire pour que cette séquence ait pu s'exécuter. L'hypothèse d'un réseau de Petri sûr étant posée, il est possible de supprimer les séquences impossibles et de ne conserver que les séquences possibles.

1.2.4.3 Automates

Une évolution sur le système peut être représentée par une trajectoire chem = $((q_0, \ldots, q_m), (l_1, \ldots, l_m))$ (présentée en annexe A) sur le modèle.

Si les états du système ne nous intéressent pas, on peut ne considérer que la *trace* d'une trajectoire, c'est-à-dire la séquence des événements.

Observations : L'hypothèse la plus simple concernant les observations est de considérer que les observations émises sont toutes reçues dans l'ordre de leur émission (hypothèse de complétude).

Le diagnostic d'un système modélisé par un automate revient à chercher sur le système toutes les trajectoires du modèle dont les événements observables correspondent aux événements observés (voir [SSL+95]). Le diagnostic de systèmes à événements discrets peut alors se voir comme une recherche de chemins dans un automate contrainte par les observations émises : $\Delta = Mod \otimes Obs$ où *Mod* est le modèle du système et *Obs*

la séquence des observations. L'ensemble des trajectoires peut être représenté par un automate.

Exemple : Reprenons l'exemple de l'automate présenté à la figure 1.4. Considérons que les événements observables sont b et p. Si le superviseur a reçu la seule observation p, alors l'automate de la figure 1.7 est le diagnostic du système.

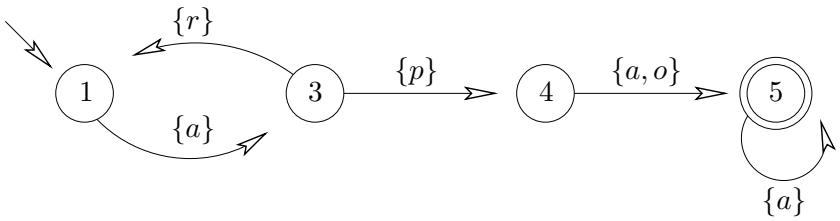

FIG. 1.7 – Exemple d'automate diagnostic

La recherche du diagnostic peut alors s'écrire comme présenté dans l'algorithme 1. Cet algorithme est utilisé par exemple dans [Pen02] pour le diagnostic local. La fonction développe_invisibles construit tous les chemins ne générant aucune observation et partant de l'ensemble d'états initiaux en paramètre, et renvoie la liste des états atteints. Cette fonction correspond à une ε-fermeture (*ε-closure*). La fonction développe_observation(es, o) construit toutes les transitions partant d'un état de es et générant l'observation o, et renvoie la liste des états atteints. L'automate est construit petit à petit pendant le développement des chemins.

Algorithme 1 Algorithme classique de recherche de trajectoires dans un modèle par automate

Recherche des trajectoires sur le modèle *Mod* **consistantes avec l'ensemble des observations** *Obs*
états_courants = développe_invisibles(I)
Tant que il reste une observation o à la liste *Obs* **faire**
  états_courants = développe_observation(états_courants, o)
  états_courants = développe_invisibles(états_courants)
Fin tant que
F_Δ = états_courants $\cap F$
return Δ

Diagnostiqueur *Observations :* L'approche diagnostiqueur considère les mêmes hypothèses que considérées précédemment, c'est-à-dire la complétude des observations.

Le diagnostiqueur (*diagnoser*) a été introduit dans [SSL+95] et [SSL+96]. Le diagnostiqueur est un automate déterministe dont les transitions ne sont étiquetées que par des observations du système. À chaque état du diagnostiqueur est associé un ensemble de couples état-panne. La sémantique du diagnostiqueur est la suivante : considérons

une séquence d'observations émise par le système, alors l'état du diagnostiqueur est l'ensemble des états possibles dans le système à l'issue de l'émission de cette séquence d'observations[1] et l'ensemble des pannes ayant eu lieu si on atteint cet état.

Prenons ainsi le modèle de la figure 1.8. Sur cette figure, aucun état n'est final. Cette notion n'a pas de sens dans un contexte en-ligne, puisqu'on fait le diagnostic alors que le système n'a pas fini de fonctionner. Aussi, lorsqu'on utilise des formalismes nécessitant de définir les états finaux, on considère que tous les états sont finaux. Sur ce modèle, les événements observables sont a, b, c et d, les événements de panne F_1 et F_2, et l'unique événement non observable est z. Le diagnostiqueur de ce modèle est présenté à la figure 1.9. Considérons le deuxième bloc de la figure étiqueté $2\ N\ 5\ P_1$. Ce bloc signifie qu'après les observations reçues pour parvenir à cet état du diagnostiqueur, l'état courant du système est soit 2 (auquel cas aucun événement de panne n'a eu lieu, N signifiant *normal*) soit 5 (auquel cas l'événement P_1 a eu lieu).

Une fois le diagnostiqueur calculé, la tâche de diagnostic est très simple. Il suffit de suivre le chemin sur le diagnostiqueur à l'aide des observations reçues (en considérant qu'elles parviennent sans perte et de manière ordonnée), et l'état du diagnostiqueur indique les états possibles du système ainsi que les événements de panne ayant pu se produire. Ainsi, une fois l'état $6\ P_1$ du diagnostiqueur atteint, il est possible de diagnostiquer que l'événement de panne P_1 a eu lieu.

[MR02] donne une méthode de construction de diagnostiqueur pour un système modélisé par Bdd.

Approches à la Lamperti, Zanella *et coll.* L'approche proposée par Lamperti, Zanella *et coll.* présentée en [LZ03b] étend les hypothèses sur les observations.

Observations : Dans cette approche, on considère qu'il y a des incertitudes et un ordre partiel sur les observations émises.

Notons tout d'abord que cette approche considère un formalisme particulier pour la représentation des événements observables et des événements de panne. En plus du modèle, le superviseur dispose de deux tables appelées *viewer* et *ruler* qui indiquent respectivement quelles transitions sont observables et quelles transitions sont fautives. Ces tables sont externes au modèle, contrairement aux précédentes modélisations. Cela permet de modifier facilement les événements observables.

Les observations sont représentées sous la forme d'un graphe direct acyclique appelé *graphe d'observations* comme présenté sur la partie gauche de la figure 1.10. Un nœud de ce graphe représente une observation. ε indique qu'aucune observation n'a été émise. Remarquons que cette observation peut être incertaine, c'est-à-dire que l'on peut ne pas connaître l'observation de manière précise (voir par exemple c_1, r qui indique qu'une observation c_1 ou une observation r a été reçue) voire ne pas être certain qu'il y a eu une observation émise (voir par exemple l, ε). Lorsqu'un chemin existe entre deux nœuds, cela signifie que l'observation du premier nœud a été émise avant l'observation du second. Par exemple, sur la figure 1.10, l'observation sh a été émise avant toutes les autres observations. En revanche, on ne sait pas si l'observation o_1 a été émise avant l,

[1] C'est-à-dire avant qu'aucun événement non observable n'ait eu lieu.

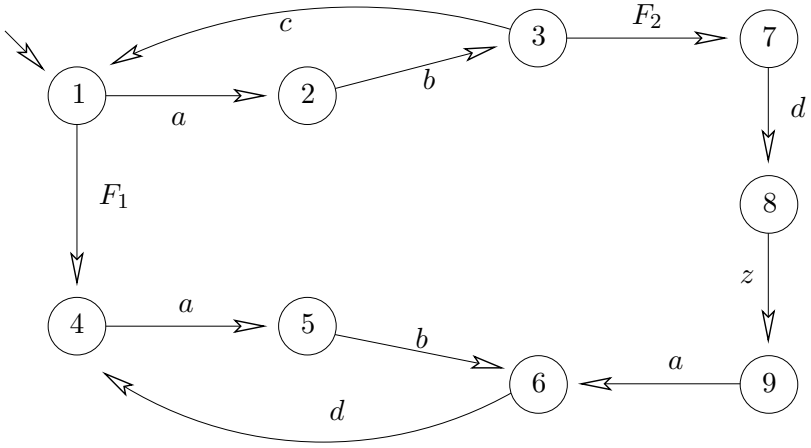

FIG. 1.8 – Exemple de modèle

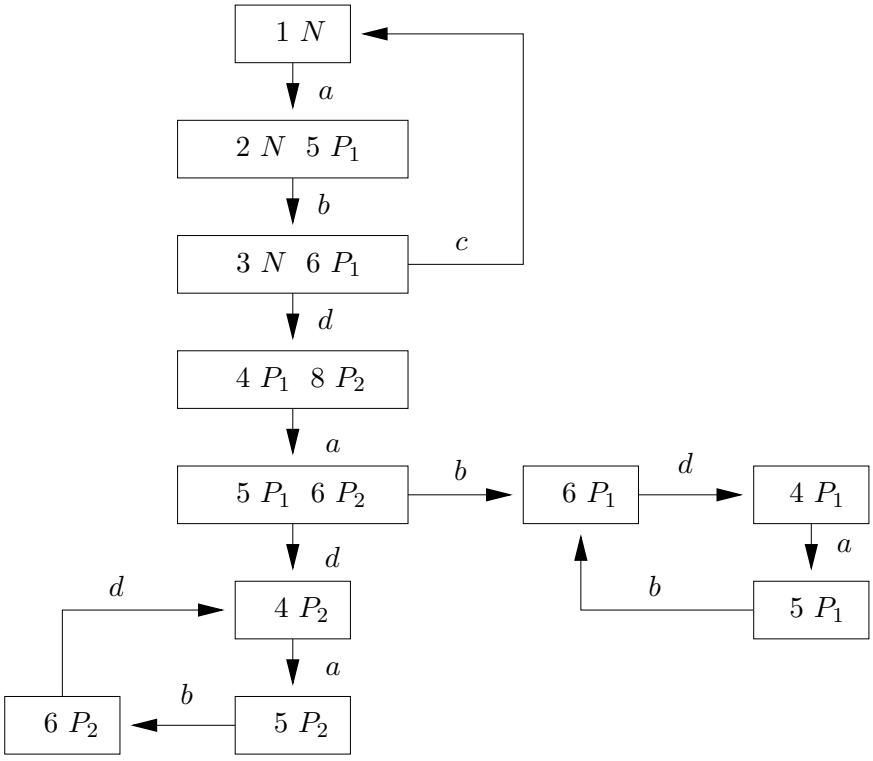

FIG. 1.9 – Diagnostiqueur du modèle de la figure 1.8

ε. Les auteurs considèrent qu'une seule observation est émise à un instant donné.

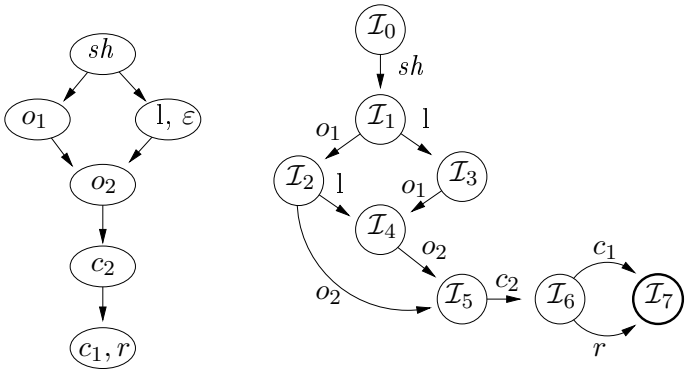

FIG. 1.10 – Graphe d'observations et espace d'index

À partir de ce graphe, est construit l'*espace d'index*. Il s'agit d'un automate déterministe et acyclique qui représente l'ensemble des séquences d'observations qui ont pu être émises. Chaque état (appelé *index*) de l'automate correspond à une ou plusieurs séquences d'observations étiquetant les différents chemins menant à cet état. L'espace d'état du graphe d'observations de la figure 1.10 est présenté à droite de la figure. Ainsi, l'index $\mathcal{I}_3$ correspond à l'émission de la séquence d'observations *sh,l*. L'index $\mathcal{I}_4$ correspond aux séquences *sh,*o_1*,l* et *sh,l,*o_1.

L'opération de base du diagnostic dans [LZ03b] est la fermeture silencieuse (*silent closure*) équivalente à la fonction développe_invisibles à la différence qu'elle part d'un unique état q. Partant d'un état du système, on construit l'automate qui représente l'ensemble des comportements possibles partant de cet état et ne générant aucune observation. Une fermeture silencieuse est donc une partie du modèle dont l'état initial est q et tous les états sont finaux. Chaque fermeture silencieuse est associée à un index de l'espace d'index.

La sémantique est la suivante : soit une fermeture silencieuse construite à partir d'un état q associée à l'index $\mathcal{I}$. Alors, il existe une séquence d'observations *seq* qui étiquette un chemin menant à $\mathcal{I}$ telle qu'un chemin sur *Mod* mène de q_0 à q, émet la séquence des observations *seq* et se termine par l'émission de la dernière observation de *seq*.

Plutôt que de donner l'algorithme utilisé dans [LZ03b], nous présentons un exemple. La figure 1.11 donne un exemple partiel pour l'espace d'index de la figure 1.10. Depuis l'état initial, on construit une fermeture silencieuse (représentée par une boite) associée à $\mathcal{I}_0$. Depuis les états de cette fermeture, il existe deux transitions étiquetées par *sh* (observation de la seule transition sortant de l'index $\mathcal{I}_0$ sur l'espace d'index) menant à q_1 et q_2. Deux nouvelles fermetures silencieuses sont ainsi calculées associées à $\mathcal{I}_1$.

Il est à présent nécessaire de considérer les transitions sur le modèle étiquetées par o_1 ou l issue d'un état de ces fermetures silencieuses. Il existe quatre transitions pour o_1 menant à deux états q_3 et q_4. On construit donc deux fermetures silencieuses F_1 et F_2. Sur l'espace d'index, l'index atteint depuis $\mathcal{I}_1$ par une transition étiquetée par o_1 est

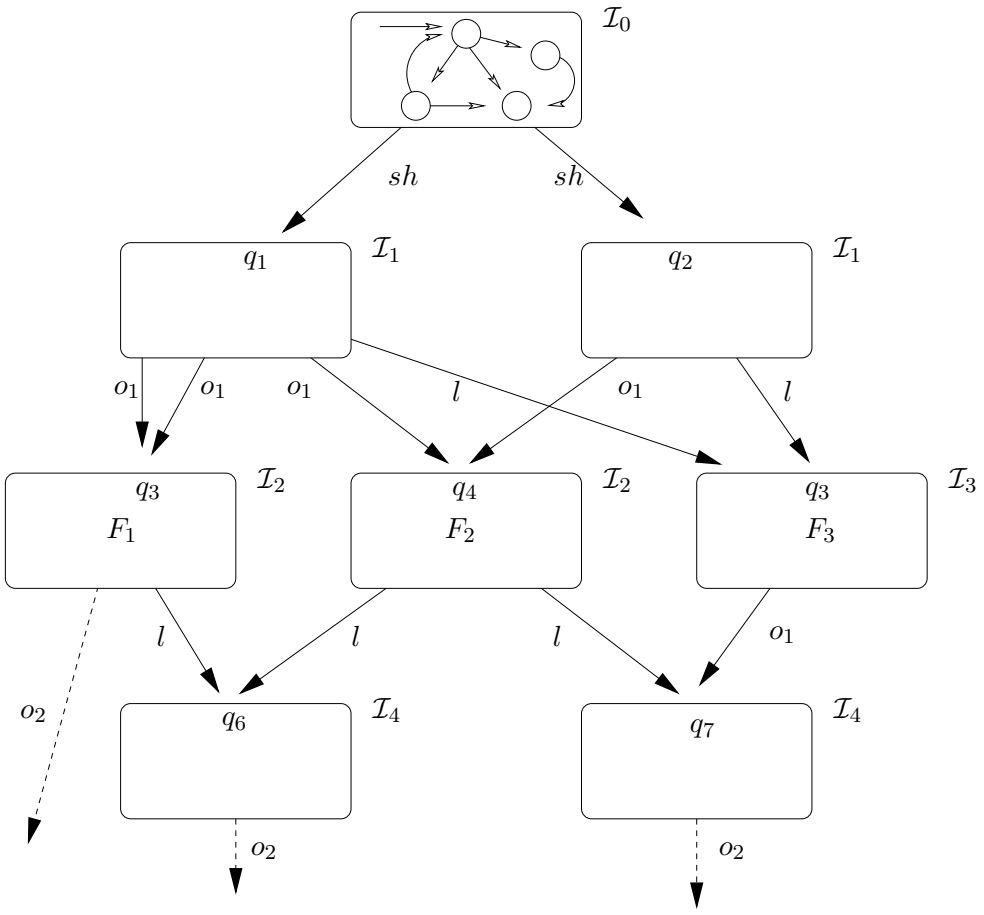

FIG. 1.11 – Exemple de diagnostic

$\mathcal{I}_2$. Ces deux fermetures F_1 et F_2 sont donc associées à $\mathcal{I}_2$. Depuis les deux fermetures associées à $\mathcal{I}_0$, il y a deux transitions pour l menant à à l'état q_3. La fermeture F_3 est générée à partir de du même état que F_1 mais est associée à $\mathcal{I}_3$. Les deux fenêtres sont bien différentes. Notons cependant qu'il est possible d'un point de vue algorithmique de ne pas calculer une deuxième fois cette fermeture.

1.2.4.4 Synthèse

Les algorithmes que nous avons présentés considèrent généralement que les observations reçues sont complètes, c'est-à-dire que toute observation émise est reçue par le superviseur. De plus, on considère généralement que les observations sont sûres.

Dans les applications réelles, ces hypothèses ne sont souvent pas respectées. Une solution est de considérer que le comportement des canaux de communication est intégré dans le modèle du système. Ainsi, les observations reçues par le superviseur sont identiques aux observations émises par le système. Cependant, intégrer les canaux de

communication au système introduit une combinatoire très importante qui n'est pas gérable dans les cas réels.

Notre but dans cette thèse est d'autoriser des hypothèses plus générales que celles généralement considérées. En particulier, nous souhaitons pouvoir considérer un ordre partiel des observations, des incertitudes sur les observations et des pertes d'observation.

Nous avons vu que les travaux de G. Lamperti et M. Zanella [LZ03b] permettaient de traiter des observations partiellement ordonnées et incertaines en utilisant un espace d'index construit à partir d'un graphe d'observations. L'approche que nous développons dans cette thèse est très semblable à celle-ci. Nous nous appuyons aussi sur les travaux de L. Console, Cl. Picardi et M. Ribaudo [CPR00] dans lesquels les auteurs représentent les observations sous la forme d'un langage, mais sans préciser toutes les possibilités de ce langage.

Les approches que nous avons présentées concernaient le diagnostic hors-ligne. Le diagnostic en-ligne implique des problèmes supplémentaires que nous développons dans la section suivante.

Nous avons précisé que notre approche s'appuyait sur un formalisme à base d'automates. Une des difficultés de ce formalisme pour les systèmes réels est que la taille du modèle est exponentielle par rapport au nombre de composants que comporte le système. Aussi, la modélisation de systèmes d'une taille suffisamment importante est impossible. L'approche décentralisée a été développée dans ce sens et nous la présentons en section 1.4.

Enfin, la problématique associée à la reconfiguration des systèmes n'a pas été beaucoup étudiée dans la littérature. Nous la présentons en section 1.5.

1.3 Diagnostic en-ligne

Jusqu'ici, nous avons considéré le diagnostic *hors-ligne* de systèmes à événements discrets. Dans ce contexte, toutes les observations devant être reçues ont été effectivement reçues, et on effectue un diagnostic *a posteriori* du système. Le diagnostic en-ligne, c'est-à-dire pendant que le système fonctionne, est plus difficile pour plusieurs raisons que nous donnons.

Nous présentons d'abord le diagnostic en-ligne, et en particulier les difficultés liées à cette tâche, puis nous montrons les algorithmes développés dans ce cadre.

1.3.1 Définition du diagnostic en-ligne

Nous présentons d'abord le diagnostic en-ligne, et nous mettons en valeur la notion de calcul incrémental du diagnostic que nous précisons par la suite.

1.3.1.1 Diagnostic en-ligne

Étant donné un flux d'observations sur le système, nous cherchons à effectuer un diagnostic du système. On qualifie ce diagnostic d'*en-ligne*.

Définition 1.9 (Diagnostic en-ligne)**.**
Le diagnostic en-ligne *consiste à diagnostiquer à différentes dates consécutives un système pendant que celui-ci émet un flux d'observations.*

Le diagnostic en-ligne consiste à calculer le diagnostic du système pendant que celui-ci fonctionne et émet des observations. Le système est censé pouvoir fonctionner pendant un temps non borné. Le but est de donner un état du fonctionnement actuel du système. Il n'est donc pas acceptable d'avoir un retard trop important. Pour cette raison, le diagnostic peut avoir du retard (notamment en cas de grande complexité du calcul) mais ce retard ne doit pas croître indéfiniment.

Le diagnostic en-ligne conduit à deux difficultés. La première difficulté est la complexité du calcul. Comme nous l'avons dit, le but est de fournir un diagnostic pendant que le système fonctionne, et donc le plus rapidement possible. De plus, le diagnostic en-ligne se fait généralement à plusieurs reprises. Si l'outil chargé du diagnostic ne parvient pas à gérer le flux d'observations et inférer le diagnostic suffisamment rapidement, alors le diagnostic en-ligne est impossible. Le calcul du diagnostic doit donc être suffisamment efficace. D'autre part, puisque le nombre d'observations augmente régulièrement, il est nécessaire de disposer d'un diagnostic *incrémental*, c'est-à-dire utilisant le diagnostic de la date t_{i-1} lors du calcul du diagnostic de la date t_i. Du point de vue incrémental, la complexité du traitement des nouvelles observations ne doit pas dépendre du nombre d'observations reçues avant t_{i-1} (puisque ce nombre augmente de manière non bornée).

La seconde difficulté porte sur le raisonnement lui-même. En effet, le diagnostic s'effectue sur un système qui génère des observations, et certaines de ces observations peuvent ne pas être encore parvenues au superviseur (non complétude des observations). Le contexte en-ligne ajoute donc des incertitudes sur les observations.

Remarquons que dans le cadre du diagnostic en-ligne, [Gue03] note l'efficacité du diagnostic selon les trois cas suivants :

- Dans le meilleur des cas, le diagnostic *prédit* que le système va tomber en panne.
- Sinon, le diagnostic peut indiquer que le système *est* actuellement en panne.
- Enfin, lorsque ces deux cas ne sont pas possibles, le diagnostic peut indiquer que le système *a été* en panne. Ceci permet de donner un diagnostic partiel de l'état actuel (quand les pannes sont permanentes, notamment) ou d'avoir une explication *a posteriori* d'un comportement observé.

1.3.1.2 Diagnostic incrémental

Lorsqu'on souhaite effectuer un diagnostic sur des fenêtres d'observations de plus en plus grandes, il est possible de réutiliser les premiers calculs pour effectuer les calculs suivants. Le calcul incrémental du diagnostic consiste à réutiliser le résultat du calcul pour des fenêtres de plus en plus grandes.

Définition 1.10 (Calcul incrémental du diagnostic)**.**
Soit $\mathcal{W}$ et $\mathcal{W}'$ deux fenêtres telles que $\mathcal{W}$ est un préfixe de $\mathcal{W}'$. Alors, le calcul incrémental *du diagnostic Δ' de $\mathcal{W}'$ est le calcul utilisant le diagnostic Δ de $\mathcal{W}$ et $(\mathcal{W}' - \mathcal{W})$.*

Le calcul incrémental du diagnostic a pour but d'accélérer le calcul de Δ' en utilisant le calcul $\mathcal{W}$ déjà effectué. La fenêtre $\mathcal{W}$ est un préfixe de $\mathcal{W}'$ et il est généralement suffisant de calculer le diagnostic de $\mathcal{W}' - \mathcal{W}$ puis de concaténer ce diagnostic au diagnostic de $\mathcal{W}$ pour obtenir le diagnostic de $\mathcal{W}'$. Une difficulté est alors de découper les fenêtres des observations.

1.3.2 Algorithmes pour le diagnostic en-ligne

Lorsque les hypothèses sur les observations sont simples (complétude des observations), le diagnostic en-ligne consiste à construire le diagnostic par les méthodes présentées hors-ligne en utilisant les observations émises, et enrichir le diagnostic à chaque fois qu'on connaît de nouvelles observations émises. Par exemple, il est possible de continuer le dépliage du réseau de Petri ou continuer à utiliser le diagnostiqueur à partir de l'état précédent atteint dans le diagnostiqueur. Nous donnons ici deux cas supplémentaires.

Diagnostic en-ligne de systèmes actifs Les travaux de Lamperti, Zanella et coll. ont étendu le problème du diagnostic pour le cas en-ligne (appelé *continuous*). Dans [LZ03a], les auteurs considèrent le cas simple où les observations parviennent dans l'ordre de leur émission. Dans ce cadre, le diagnostic peut être obtenu de la manière suivante : tout d'abord, le diagnostic est la fermeture silencieuse de l'état initial. Lors de la réception d'une observation, on calcule toutes les transitions sortant de la fermeture et étiquetée par l'observation. Pour chaque état atteint par une de ces transitions, on calcule la fermeture silencieuse et ainsi de suite, comme présenté en introduction de cette sous-section.

Les auteurs ont montré dans [LZ04] comment généraliser ce résultat au cas des observations incertaines et partiellement ordonnées. Sans rentrer dans les détails, la méthode est proche de celle utilisée dans le cadre hors-ligne. Le graphe des observations est construit ainsi que l'espace d'index et le diagnostic selon cet espace. À la réception d'une nouvelle observation, le graphe et l'espace d'index sont mis à jour (ajout d'un nœud au graphe et d'un nombre petit[2] d'index). On peut donc considérer que la fenêtre d'observation est réduite à une seule observation si on considère le graphe des observations, ou au *morceau* d'automate ajouté à l'espace d'index. Le diagnostic est également mis à jour pour tenir compte des ajouts dans l'espace d'index.

Le diagnostic est effectué en considérant que toutes les observations nécessaires au diagnostic ont été reçues. Cependant, il est possible qu'une observation o émise avant la dernière observation o' reçue n'ait pas encore été reçue. L'algorithme permet de prendre en compte cette possibilité, mais le diagnostic en-ligne effectué après la réception de o' ne prenait pas en compte l'émission de o, et n'est donc pas *correct*. On peut qualifier ce diagnostic de *pessimiste* dans le sens où il considère qu'aucune autre observation, émise avant la dernière observation reçue, ne peut être reçue.

[2] La valeur de ce nombre dépend du type de système et de canaux de communication, mais on peut généralement considérer qu'elle est bornée et petite.

Diagnostic en-ligne par fenêtres temporelles sûres L'approche développée par Y. Pencolé, M.-O. Cordier et coll. [PCR01], [Pen02], [PC05][3] considère généralement que la période d'observation peut être découpée en fenêtres temporelles *sûres*. Une fenêtre temporelle est définie comme sûre si elle est délimitée par deux points d'arrêts *sains*, c'est-à-dire pour lesquels on dispose de la liste des observations émises à cette date. Un tel point est obtenu quand aucune observation n'est reçue après un délai qui correspond au temps maximum que nécessite une observation émise pour être reçue. Ainsi, le superviseur connait précisément les observations émises pendant chaque fenêtre et il est possible d'effectuer le calcul comme dans un contexte hors-ligne.

Il n'est pas toujours possible de construire des fenêtres sûres, et les auteurs ont proposé un diagnostic *étendu* permettant de considérer jusqu'à k observations émises pour chaque composant du système et non reçues.

1.3.3 Synthèse

Le diagnostic en-ligne pose des difficultés supplémentaires par rapport au diagnostic hors-ligne. La première difficulté est que le calcul doit être effectué de manière efficace, et donc de manière incrémentale. La seconde difficulté est de considérer des hypothèses supplémentaires sur les observations émises, à savoir que certaines observations émises peuvent ne pas avoir été reçues, et ne sont donc pas connues.

1.4 Modélisation et diagnostic décentralisés et distribués

Les approches décentralisée et distribuée voient le système comme un ensemble de parties (qu'on appelle par la suite *composants*) interconnectés. Le raisonnement est effectué de manière locale avant de fusionner ces raisonnements. On considère généralement dans la littérature que les approches décentralisées fournissent un diagnostic global pour un unique superviseur. En revanche, dans le contexte distribué, il existe plusieurs superviseurs chacun en charge d'une partie du système. Dans ce second cas, le diagnostic est donc local à chaque partie du système (voir par exemple [WYL04]).

Nous donnons d'abord les raisons pour lesquelles les approches distribuée et décentralisée ont été développées. Nous présentons ensuite les modélisations associées et les algorithmes mis en place pour bénéficier de la modélisation distribuée.

1.4.1 Justifications des approches distribuée et décentralisée

La plupart des systèmes réels que l'on cherche à diagnostiquer sont décentralisés par nature. En effet, ils sont constitués de composants ou pièces interreliés. On peut ainsi considérer les exemples de réseaux, de véhicules automobiles, de chaînes de production en usine ou de *web-services*. La conception et la construction d'un système par composants offre de nombreux avantages : les pièces sont normalisées et utilisables dans divers contextes ; elles peuvent être produites en série. Les systèmes peuvent également évoluer

[3] Cette approche est faite dans un contexte de diagnostic décentralisé expliqué en sous-section 1.4.3.2.

par la modification des composants (voir la reconfiguration en section 1.5). Une modification du système est aisée et moins coûteuse. Puisque les systèmes sont décentralisés par nature, il paraît logique de les représenter et les diagnostiquer de la même manière.

Pour les gros systèmes, la construction du modèle du système est une tâche difficile. En revanche, la modélisation de chaque composant est simple (lorsqu'on considère des composants suffisamment petits). Ainsi, modéliser le système par ses composants plutôt que globalement est beaucoup plus rapide et source de moins d'erreurs. Dans certains cas, notamment lors d'une modélisation par automate, la taille du modèle global peut être exponentielle par rapport au nombre de composants. Ainsi, il est impossible de modéliser un système comprenant ne sera-ce qu'une trentaine de composants. En revanche, la taille du modèle décentralisé est linéaire par rapport au nombre de composants.

À partir d'un modèle décentralisé, il est nécessaire d'adapter les algorithmes de diagnostic pour cette représentation. Certaines approches que nous présentons par la suite gardent une modélisation globale du système, mais effectuent néanmoins un calcul décentralisé dans le but de diminuer la complexité de ce calcul. Le calcul est effectué de manière locale aux composants, avant qu'une fusion de diagnostics soit nécessaire pour un diagnostic global.

Dans l'approche distribuée, il n'y a pas un unique superviseur, mais un ensemble de superviseurs, chacun en charge d'une partie du système. La tâche du diagnostic est donc effectuée de manière locale, et les superviseurs doivent ensuite effectuer un recoupement de leurs hypothèses avec leurs superviseurs voisins.

Notons que le choix d'une approche décentralisée ou distribuée est complètement dépendant des contraintes architecturales ou de l'utilisation qu'on souhaite faire du diagnostic. Par exemple, pour un système dont on souhaite contrôler le comportement de manière globale, on choisit un diagnostic décentralisé. En revanche, pour un contrôle local par agents, il faut une approche distribuée.

Nous présentons par la suite comment construire le modèle décentralisé du système. Puis nous présentons les algorithmes décentralisés et, dans un moindre mesure, distribués de la littérature.

1.4.2 Modélisations décentralisées

Nous considérons tout d'abord les différentes manières de modéliser un système de manière décentralisée. Les modélisations décentralisées considèrent le système comme un ensemble de composants intercommunicants. Ces communications peuvent être synchrones ou asynchrones. Nous étudions ici les modélisations à base d'algèbres de processus, de réseaux de Petri et d'automates.

1.4.2.1 Algèbres de processus

Les algèbres de processus permettent de modéliser le comportement du système de manière décentralisée.

Dans PEPA, il faut utiliser l'opération de coopération $\underset{L}{\bowtie}$. Le comportement du système peut se décrire par : $S \stackrel{def}{=} S_1 \underset{L}{\bowtie} S_2$ où S_1 est le modèle d'un sous-système, S_2

est le modèle du reste du système et L synchronise les comportements de S_1 et S_2.

Cette notation permet de représenter un système de manière hiérarchique ou comme un ensemble de composants. Les communications entre les composants sont *synchrones*.

1.4.2.2 Réseaux de Petri

Dans les réseaux de Petri, la modélisation est déjà très proche d'une modélisation décentralisée. En effet, on peut considérer qu'un composant est modélisé par un ensemble de places ainsi que les transitions menant d'un ensemble à un autre. La figure 1.12 donne un exemple de modélisation décentralisée pour le système. On voit que les places P_5 et P_9 appartiennent aux deux modèles. On appelle ces places des places d'*entrée* ou de *sortie*. Ces places permettent de représenter les communications entre les composants. Ainsi, l'émission d'un message du composant inférieur sur la figure 1.12 vers le composant supérieur se fait soit par la transition t_5 soit par la transition t_7 qui conduisent toutes les deux à la production d'un jeton dans la place P_5. La réception du message est effectuée par la consommation du jeton de cette place (ici par la transition t_9). On voit que l'émission et la réception de la communication entre les deux composants ne sont pas simultanées. En particulier, d'autres transitions peuvent être tirées entre ces deux transitions. Il s'agit donc d'une communication *asynchrone*.

1.4.2.3 Automates communicants

Les automates communicants sont également appelés *transducteurs* ([AU72]). Ces automates décrivent le comportement de chaque composant lors de la réception d'un message venant de l'environnement ou d'un autre composant, et décrivent comment ces composants émettent à leur tour des messages. Nous présentons ici le formalisme utilisé dans [Roz97] et utilisé celui dans [Pen02] qui sont à la base du formalisme utilisé dans cette thèse (présenté en section 7.2).

Dans [Roz97], un composant est modélisé par un tuple $(Q, q_0, \mathcal{E}, \delta, \mathcal{V}, \mathcal{A})$ où :
- Q est l'ensemble des états et $q_0 \in Q$ est l'état initial,
- $\mathcal{E}$ est un ensemble d'événements qui se partitionne en $\mathcal{E}_r$ (événements reçus), $\mathcal{E}_e$ (événements émis) et $\mathcal{E}_{\text{int}}$ (événements internes),
- δ est un ensemble de transitions,
- $\mathcal{V}$ est un ensemble de variables et $\mathcal{A}(\mathcal{V})$ un ensemble d'actions possibles sur $\mathcal{V}$.

Un événement $e \in \mathcal{E}$ est un couple (m, p) où m est un message et p un port. Un port est une interface avec l'extérieur. On distingue l'ensemble des ports d'entrée $\mathcal{P}_r$ (r pour réception) et l'ensemble des ports de sortie $\mathcal{P}_e$ (e pour émission). Une transition $\delta = (q_d, e_r, c, q_a, \{em\}, a)$ fait passer de l'état q_d à l'état q_a si la condition c est vraie et si l'automate lit le symbole d'entrée e_r. Dans ce cas, l'automate émet les messages de la liste $\{em\}$ et effectue l'action a.

Un exemple de transition est donné sur la figure 1.13. La transition correspondante est t =(es,(arret(tp),reset),c,hs,{(reinit,service)},tpan:=t). Le composant étant dans l'état es, si la condition c est vérifiée, et si le composant reçoit le message arret(tp)

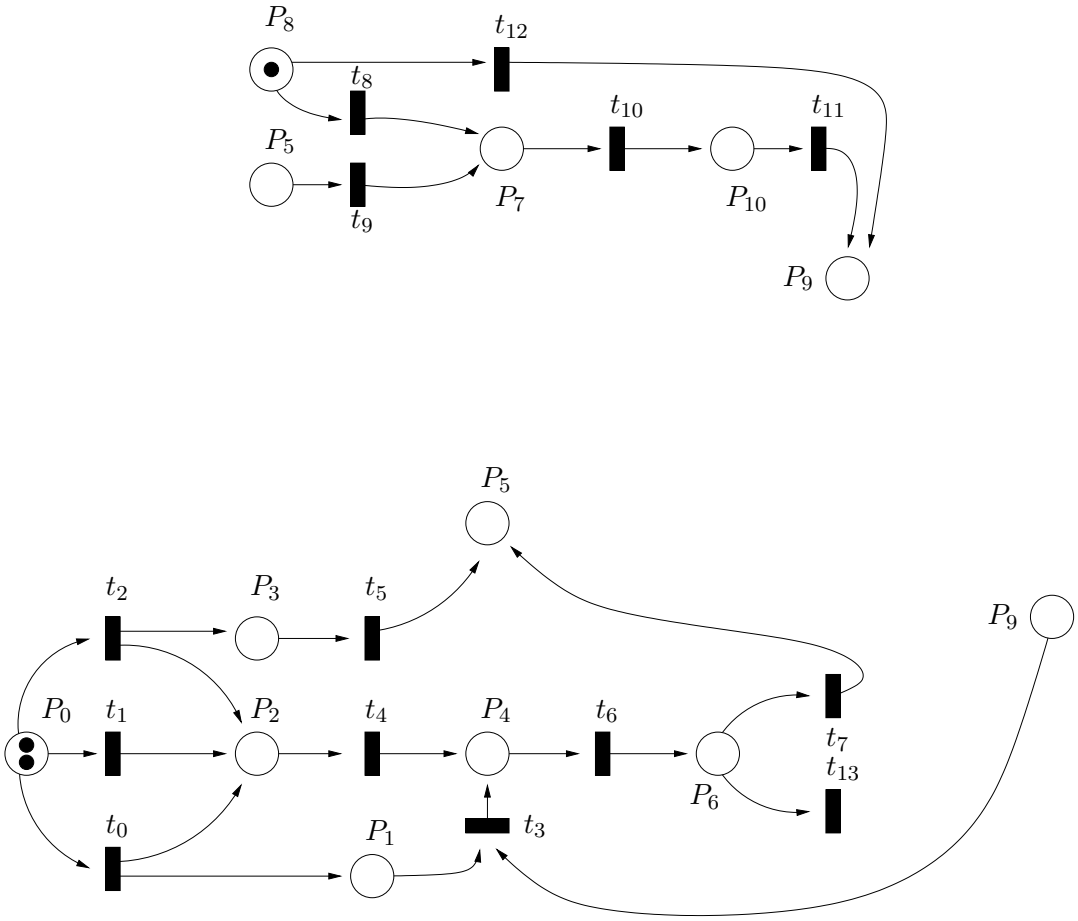

FIG. 1.12 – Modèle décentralisé du système modélisé à la figure 1.3

sur le port reset, alors le composant passe dans l'état hs et émet l'unique message reinit sur le port service tandis que la variable tpan prend la valeur de t. Sur la représentation graphique, le + représente l'émission du message et le - représente la réception.

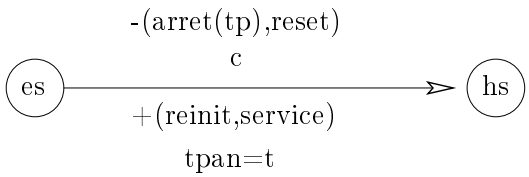

FIG. 1.13 – Exemple de transition

Le modèle du système comprend également un ensemble de *connexions*. Une connexion relie le port de sortie d'un composant et le port d'entrée d'un autre composant. Lorsqu'une connexion existe entre deux composants, alors l'émission d'un message

par le premier composant sur son port de sortie conduit à la réception *synchrone* du message par le deuxième composant sur son port d'entrée.

À l'aide de ce modèle décentralisé du système, il est possible de reconstruire le modèle du système. Considérons $(Q_1, q_{01}, \mathcal{E}_1, \delta_1, \mathcal{V}_1, \mathcal{A}_1)$ et $(Q_2, q_{02}, \mathcal{E}_2, \delta_2, \mathcal{V}_2, \mathcal{A}_2)$ les modèles de deux composants. Alors, l'automate A du sous-système constitué des deux composants est $(Q, q_0, \mathcal{E}, \delta, \mathcal{V}, \mathcal{A})$ défini ainsi :

- $Q = Q_1 \times Q_2$,
- $q_0 = (q_{01}, q_{02})$,
- $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2$,
- $\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2$,
- $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2$.

L'ensemble des transitions est calculé de la manière suivante :

Pour les transitions (q_i, e, c, q'_i, l, a) qui ne concernent que le composant i, c'est-à-dire telles qu'aucun message n'est émis sur un port connecté à un port du composant $j \neq i$, alors il existe une transition $((q_i, q_j), e, c, (q'_i, q_j), l, a)$ dans δ pour tous les états q_j du composant j.

Pour les transitions $(q_i, e_i, c_i, q'_i, l_i, a_i)$ et $(q_j, e_j, c_j, q'_j, l_j, a_j)$ concernant les deux composants, c'est-à-dire telles qu'il existe un message m émis par le composant i sur un port connecté à un port du composant j, alors la transition $((q_i, q_j), e_i, c_i \cup c_j, (q'_i, q'_j), l, a_i \cup a_j)$ est ajouté à δ si e_j correspond à la réception du message m sur le port connecté. L'étiquette l correspond à l'union $l_1 \cup l_2$ où on remplace les événements d'émission et de réception de m par un événement interne. Cette opération est appelée *synchronisation* des deux transitions.

Cette modélisation permet d'obtenir le modèle global du système à partir d'une modélisation décentralisée.

La modélisation proposée dans [Pen02, PC05] reprend les mêmes concepts que la modélisation précédente. Un composant élémentaire est modélisé par un transducteur $\Gamma_i = (\Sigma^i_{dec}, \Sigma^i_{émis}, Q_i, E_i)$ où :

- Σ^i_{dec} est l'ensemble des événements déclencheurs (événements dont la cible est Γ_i),
- $\Sigma^i_{émis}$ est l'ensemble des événements émis par le composant (événements dont l'origine est Γ_i) avec $\Sigma^i_{dec} \cap \Sigma^i_{émis} = \emptyset$,
- Q_i est l'ensemble des états du composant,
- $E_i \subseteq (Q_i \times \Sigma^i_{dec} \times 2^{\Sigma^i_{émis}} \times Q_i)$ est l'ensemble des transitions.

Une transition $(q, dec, \mathcal{E}, q')$ représente le comportement du composant à la réception du message dec : il passe de l'état q à l'état q' en émettant les messages $\mathcal{E}$. Notons que les variables sont abstraites. En outre, les ports sont également abstraits. En effet, l'événement émis e par le composant Γ_i est reçu (sauf en cas d'observation) par un unique autre composant Γ_j : $e \in \Sigma^j_{déc}$.

Le modèle décentralisé du système se définit alors comme l'ensemble des modèles de chaque composant : $\Gamma = \{\Gamma_1, \ldots, \Gamma_n\}$. Les connexions entre composants sont implicites.

Le comportement global du système $\| \Gamma \|$ peut être calculé par synchronisation de manière comparable à [Roz97], mais n'est pas fait contrairement à [Roz97] pour éviter l'explosion combinatoire.

[MR02] indique que dans la modélisation par BDD ou par $p-$DD, on peut considérer chaque variable comme le modèle d'un composant virtuel. En effet, le modèle global s'obtient alors par synchronisation des modèles locaux représentés par la projection du modèle global sur chaque variable. Une modélisation par BDD peut donc, de ce point de vue, être considérée comme une modélisation décentralisée.

1.4.2.4 Modélisation à la Lamperti-Zanella

Les systèmes actifs [LZ03b] considérés par G. Lamperti et M. Zanella sont des systèmes hiérarchiques constitués de composants interreliés par des connexions asynchrones ou synchrones appelées *links*.

Chaque composant dispose d'un ensemble de *terminaux*, qui sont les points de connexion entre le composant et son environnement. Par ces terminaux entrent (dans le cas des terminaux d'entrée) ou sortent (dans le cas des terminaux de sortie) des événements. On distingue l'entrée standard *In* (pour les événements provenant de l'extérieur du système), la sortie standard *Out* (pour les événements émis vers l'extérieur du système) et la sortie d'erreur *Flt* modélisant les comportements fautifs. Formellement, le comportement d'un composant est un automate $(S, E_{\text{in}}, I, E_{\text{out}}, O, T)$ où S est un ensemble d'états, E_{in} (resp. E_{out}) l'ensemble des événements reçus (resp. émis), I (resp. O) l'ensemble des terminaux d'entrée (resp. de sortie) et $T : S \times E_{\text{in}} \times I \times 2^{E_{\text{out}} \times O} \mapsto 2^S$ l'ensemble des transitions. Une transition est empruntée lors de la réception d'un événement d'entrée reçu sur un terminal d'entrée (événement noté α), et produit un ensemble d'événements émis chacun sur un terminal de sortie différent (événements β). Par ailleurs, l'état du composant évolue pendant le franchissement de la transition. Remarquons que la transition n'est pas nécessairement déterministe.

Le modèle d'un composant est présenté à la figure 1.14. Dans cet exemple, le composant comporte deux terminaux d'entrée, I_l et I_r représentés par les triangles, et quatre terminaux de sortie, O_l, O_r, O_1 et O_2 représentés par les cercles. Ces terminaux sont en plus des terminaux standard. Prenons l'exemple de la transition $\mathbf{T}_3$. Cette transition peut s'expliquer de la manière suivante. Le composant étant dans l'état **Normal**, la réception de l'événement z_2 provenant du terminal I_l conduit à l'émission du message z_2 sur le terminal O_2. Par ailleurs, le composant passe dans l'état **Shl**.

Le système est constitué d'un ensemble de composants interreliés au travers de connexions appelées *links*. Un lien est un 5-tuple (Y, I, O, χ, P) où $Y \in \{SYNC, ASYNC\}$ indique si la connexion est synchrone, I (resp. O) correspond au terminal d'entrée (resp. de sortie) de la connexion, χ est la capacité d'événements pouvant rester en attente dans la connexion et P indique la politique de saturation. Par exemple, si un événement est transmis dans une connexion ayant atteint sa limite de capacité χ, alors ce dernier événement peut être perdu, ou il peut remplacer le dernier message. Les connexions de type asynchrone sont considérées comme des files *FIFO* (*first in, first out*).

Le système comporte alors deux types de comportements distincts. D'une part, le système peut évoluer par l'occurrence d'un événement extérieur (c'est-à-dire *via* le terminal *In* d'un composant). D'autre part, si au moins un lien dispose d'un événement

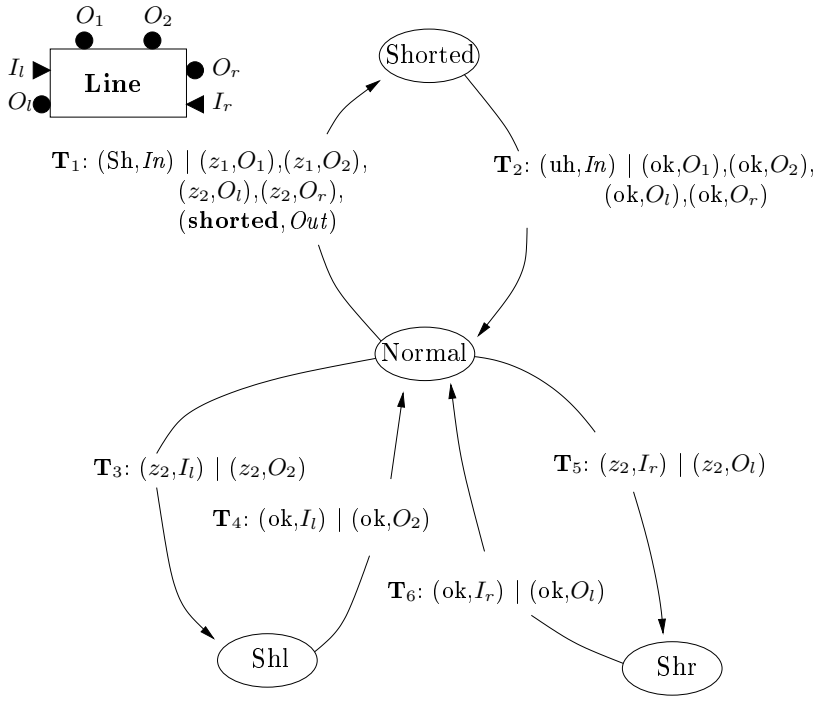

FIG. 1.14 – Exemple d'un modèle de composant

non transmis, c'est-à-dire toujours dans la file d'attente des messages, alors le système est en mode réactif (*reacting mode*) et l'événement peut être transmis conduisant à l'évolution du composant récepteur. Si les liens ont transmis la totalité des événements, alors le système passe dans le mode passif (*quiescent mode*). À chaque fois, les composants voisins peuvent évoluer de manière synchrone en cas d'émission d'événement vers un lien synchrone.

1.4.3 Algorithmes décentralisés

Les algorithmes de diagnostic décentralisés raisonnent de manière locale sur les composants et fusionnent les résultats de ces raisonnements pour obtenir un diagnostic global.

1.4.3.1 Diagnostic décentralisé par protocole

Il existe des approches *par protocole* pour le diagnostic décentralisé [DLT98, WYL04, SW00]. Ces approches s'appuient sur le modèle global du système. Dans ces approches, on considère qu'il existe plusieurs *sites*. Chacun de ces sites reçoit certaines observations (par exemple, le site 1 reçoit les observations émises par les capteurs A et B). Plusieurs sites peuvent recevoir les mêmes observations.

Localement à chaque site, est effectué un raisonnement à partir des observations reçues. Ce raisonnement est généralement effectué par un diagnostiqueur construit localement au site. Ainsi, la complexité du modèle global n'est pas un problème pour le raisonnement lors du diagnostic. En revanche, elle l'est lors du calcul des diagnostiqueurs locaux. De manière locale, les diagnostiqueurs obtiennent un ensemble de pannes qu'ils fournissent à un *coordinateur* qui a la charge de construire le diagnostic global.

Le coordinateur s'appuie sur des notions de diagnosticabilité locale pour déterminer le diagnostic global. Ainsi, les auteurs définissent la F-codiagnosticabilité, la NF-codiagnosticabilité et la diagnosticabilité indépendante (voir [Sen98]).

Il est alors possible d'utiliser ces propriétés et d'utiliser un protocole pour déterminer l'occurrence d'événements de panne. Chaque site peut émettre un message au coordinateur du type : *faute*, *pas de faute*, *faute si aucun site ne dit « pas de faute »*, *pas de faute si aucun site ne dit « faute »*, *rien*. [WYL04] donne la table utilisée par le coordinateur avec notamment les conflits de diagnostic possibles.

1.4.3.2 Diagnostic décentralisé à la Pencolé–Cordier

L'approche proposée par Y. Pencolé et M.-O. Cordier et résumée dans [PC05] utilise la modélisation décentralisée du système pour construire le diagnostic. Le principe est de considérer que le modèle du système est de taille très importante, tandis que le diagnostic en lui-même est relativement petit. La raison de cette propriété est que les observations sur le système contraignent fortement les comportements possibles du système.

Aussi, l'approche décentralisée propose de construire un diagnostic *local* à chaque composant puis de *fusionner* les diagnostics selon une certaine *stratégie* pour produire le diagnostic global du système. On peut représenter cette méthode par la figure 1.15.

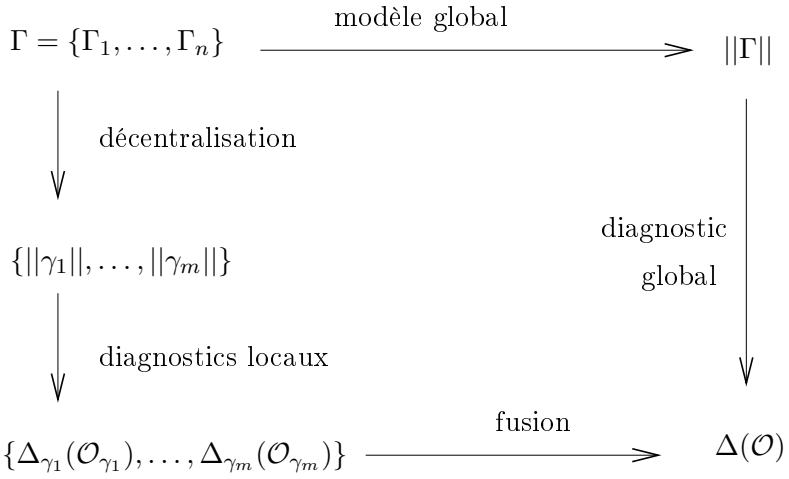

FIG. 1.15 – Approche centralisée / approche décentralisée

Le diagnostic local Δ_γ à un composant γ est un calcul du diagnostic (représenté sous forme d'automate) prenant en compte le modèle du composant et les observations

effectuées sur ce composant. Aucune restriction n'est effectuée dans les communications entre le composant et le reste du système. Ainsi, certaines trajectoires du diagnostic sont possibles sous réserve que les communications avec le reste du système qu'elles comportent soient possibles. C'est ce qui est désigné sous le terme d'*hypothèse de communication*.

Une fois les diagnostics locaux calculés, il faut fusionner les diagnostics. Ainsi, soit γ un sous-système constitué des sous-systèmes γ_1 et γ_2. Alors, à partir des comportements de γ_1 et γ_2, on calcule les comportements possibles de γ. Ces comportements sont beaucoup plus faciles à calculer à partir des diagnostics Δ_{γ_i} puisque le nombre de comportements possibles est restreint. La fusion de deux diagnostics permet de vérifier les hypothèses qui étaient effectuées sur les communications entre les deux sous-systèmes ; en effet, il faut vérifier que tout message envoyé par un composant du premier sous-système à un composant du second sous-système a bien été reçu, et réciproquement. En revanche, les hypothèses de communication entre γ et le reste du système ne sont pas vérifiées à ce niveau du calcul.

La fusion des diagnostics est effectuée deux à deux pour restreindre autant que possible le nombre de comportements à étudier. On appelle *stratégie* l'ordre dans lequel la fusion des diagnostics s'effectue. Il apparaît que certaines stratégies sont plus intéressantes que d'autres. En particulier, il est préférable de fusionner en priorité les diagnostics de sous-systèmes dont les comportements sont dépendants, pour restreindre les comportements (et vérifier la non validité de certaines hypothèses) et réduire la taille du diagnostic. Il est possible de se baser sur le modèle pour savoir quels sous-systèmes communiquent le plus entre eux, et d'avoir ainsi une stratégie de fusion statique. Cependant, d'une exécution à une autre, ce ne sont pas les même sous-systèmes qui communiquent entre eux. Les auteurs ont donc proposé une stratégie *dynamique* où l'ordre de fusion est choisi de manière dynamique en se basant sur les diagnostics et non sur le modèle décentralisé. Dans certains cas, la fusion des diagnostics de deux sous-systèmes ne fournit aucune information supplémentaire. L'utilisation d'une stratégie dynamique permet de profiter de ce phénomène en conservant les diagnostics locaux aux sous-systèmes ne communiquant pas deux à deux.

Une difficulté qui apparaît lorsque l'on fusionne les diagnostics est le problème de *concurrence*. La concurrence est le phénomène représenté par l'occurrence d'événements sur des sous-systèmes disjoints. Le problème de ce genre de comportement est que le nombre de séquences représentant des comportements concurrents est factoriel, et le nombre d'états nécessaires pour leur représentation est exponentiel par rapport au nombre d'événements concurrents.

Les travaux développés par Y. Pencolé et M.-O. Cordier utilisent une notion proche de la concurrence : l'*indépendance*. Dans l'indépendance, l'ordre d'occurrence de certaines paires d'événements ne change pas le comportement global du système. Formellement, soit a et b deux étiquettes de transitions. Alors, si pour tout état q :

- si a peut avoir lieu en q et q' est l'état du système après l'occurrence de a, alors b peut avoir lieu en q si et seulement si b peut avoir lieu en q',
- réciproquement pour b et a,

– si a et b peuvent avoir lieu en q, alors l'état atteint depuis q après les occurrences de a puis b est le même que celui atteint après les occurrences de b puis a.

Il est alors possible de représenter des comportements indépendants par des *traces de Mazurkiewicz* [Maz88]. Considérons par exemple les événements a, b et c tels que a et b sont indépendants ainsi que b et c (mais pas a et c). Alors, la trace de Mazurkiewicz a, b, c représente non seulement la séquence a, b, c mais également les séquences b, a, c et a, c, b, c'est-à-dire toute séquence qui peut être obtenue par permutations successives de deux événements indépendants se suivant dans la séquence d'événements.

Y. Pencolé et M.-O. Cordier utilisent les techniques de réduction d'ordre partiel [Pel93] utilisant les traces de Mazurkiewicz pour réduire la taille des diagnostics de (sous-)systèmes.

Nous avons présenté dans [CGLP03a] et [CGLP03b] une propriété proche de l'indépendance appelée l'*interversibilité*. Cette propriété indique que deux événements peuvent être échangés dans une séquence lorsqu'ils sont séparés par certaines séquences d'événements. L'adaptation de la technique de réduction d'ordre partiel à l'interversibilité permet de réduire la taille de l'arbre de recherche dans le cadre du diagnostic ou de la planification.

1.4.3.3 Diagnostic décentralisé à la Lamperti–Zanella

Le diagnostic de systèmes actifs permet également un diagnostic décentralisé (appelé en l'occurrence *diagnostic modulaire*). Cette approche est très semblable à celle développée par Pencolé *et al.*

Le système est divisé de manière hiérarchique en *clusters*. Un cluster ξ est modélisé par $M_\xi = (\mathbb{C}_\xi, \mathbb{L}_\xi, \mathbb{D}_\xi)$, c'est-à-dire un ensemble de composants $\mathbb{C}_\xi$ appelés nœuds, l'ensemble de liens $\mathbb{L}_\xi$ entre les composants et l'ensemble de liens connectés avec l'extérieur $\mathbb{D}_\xi$.

La décomposition d'un (sous-)système ξ est un ensemble de clusters disjoints : $\Xi(\xi) = \{\xi_1, \ldots, \xi_n\}$. L'interface $\mathit{Interf}(\Xi(\xi))$ de la décomposition $\Xi(\xi)$ est l'ensemble des liens entre deux clusters.

À nouveau, un cluster peut être décomposé en clusters de taille plus petite. Un graphe dit de *reconstruction* représente la décomposition hiérarchique du système.

Sans entrer dans les détails, le diagnostic est effectué de manière locale, et éventuellement parallèlement, pour chaque cluster. Ensuite, il est possible de *reconstruire* le diagnostic d'un cluster de plus haut niveau à partir des diagnostics de chacun des clusters de la décomposition du cluster. Remarquons cependant que le choix de fusion des diagnostics au sein d'un cluster ne se fait pas de manière dynamique, mais est régi par la structure hiérarchique modélisant le système.

1.4.4 Algorithmes distribués

Nous avons vu que le diagnostic distribué consiste à calculer les diagnostics locaux de sous-systèmes. Le diagnostic global du système n'est pas calculé, et il n'y a pas de superviseur global du système.

Notre thèse ne se situe pas dans ce contexte. Aussi, nous présentons les approches distribués de manière rapide.

1.4.4.1 Diagnostic distribué par réseau de Petri

Le diagnostic distribué par réseau de Petri [BHFJ04] cherche à calculer un dépliage pour chaque composant, par exemple les deux composants de la figure 1.12. Dans le réseau du premier composant, le marquage initial n'inclut pas de jeton pour la place P_5 et aucune transition ne fournit de jeton à la place P_5. Cependant, il s'agit d'une place d'*entrée* et les jetons de cette place correspondent à des messages envoyés par le second composant. Aussi, d'un point de vue local, le dépliage du réseau du premier composant peut comporter autant de places étiquetées par P_5 que l'on veut.

Une place d'entrée dans un dépliage de diagnostic local doit correspondre à une place de sortie dans le dépliage d'un autre diagnostic local. Pour cela, les diagnostics sont fusionnés deux à deux, jusqu'à obtenir une stabilisation des diagnostics. On peut se référer à [BHFJ04] pour plus de précisions.

1.4.4.2 Diagnostic de réseau de Petri par retour arrière

Dans [JB05], les auteurs ont considéré l'extension de l'approche par retour arrière que nous avons présentée en fin de sous-section 1.2.4.2, page 21 pour permettre un diagnostic distribué. Dans cette approche, le diagnostic consiste à vérifier par retour arrière le marquage minimum nécessaire pour expliquer les observations reçues. Lorsque le marquage minimum indique des jetons sur des places d'entrée, l'outil de diagnostic local transmet cette information à l'outil chargé du diagnostic du composant transmettant des messages *via* cette place. Alors, le superviseur local cherche à expliquer l'émission de ces messages (en demandant éventuellement de manière récursive que soient expliquées des réceptions de messages par son propre composant) et indique si le composant a pu transmettre les messages. Un circuit entre une place d'entrée et une place de sortie doit comporter au moins une observation pour éviter les boucles infinies dans le processus.

1.4.4.3 Diagnostic distribué par langage

Dans [Sen98], l'auteur considère que le comportement du système peut être modélisé par un langage L sur l'ensemble de mots Σ. Le diagnostic est donc un langage. L'hypothèse considérée dans cet article est que le comportement est modélisé de manière globale.

R. Su et W. Wonham [SW04] reprennent un formalisme comparable. Le diagnostic local est représenté par le langage L_i projection de L sur l'ensemble Σ_i. Un lien virtuel synchrone existe entre deux sites i et j si ces sites partagent certains événements $\Sigma_i \cap \Sigma_j \neq \emptyset$. Les liens sont arbitrairement orientés, donnant une relation de parenté et de filiation entre les sites (si possible pour produire un arbre).

Une fois le diagnostic local calculé, l'algorithme CPLC de diagnostic consiste à synchroniser un diagnostic local d'un site avec les diagnostics d'une partie de ses voisins puis à reprojeter le comportement sur l'ensemble des événements Σ_i du site. Cette

synchronisation se passe en deux boucles, tout d'abord des sites fils aux sites parents (des feuilles à la racine) et ensuite dans l'autre sens. Pour des langages finis, l'algorithme termine. De plus, pour un graphe sous forme d'arbre, l'algorithme nécessite au plus deux boucles.

Les auteurs proposent également un second algorithme nommé CPGC. Cet algorithme est hiérarchique et consiste à partitionner l'ensemble des sites en sous-systèmes[4]. Le langage de chaque sous-système est calculé (synchronisation des langages de chaque site) et éventuellement abstrait. Sur ce second niveau, l'algorithme CPLC est alors appliqué. Le diagnostic de chaque sous-système est ensuite projeté sur chaque site, ce qui permet une convergence plus rapide avant l'application de CPLC sur l'ensemble des sites.

1.4.5 Synthèse

Les approches décentralisées sont indispensables pour pouvoir considérer des systèmes constitués d'un grand nombre de composants. La littérature comprend plusieurs raisons pour privilégier un calcul décentralisé à un calcul global du diagnostic, en particulier de performances meilleures dans la mesure où un raisonnement local préalable (et éventuellement en parallèle pour chaque composant) permet de réduire fortement l'espace de recherche pour le diagnostic. Cependant, nous considérons que la principale raison pour laquelle l'approche décentralisée est indispensable est que le modèle global (au moins dans une approche par automate) est impossible à construire pour un système d'une taille suffisamment importante.

Comme nous l'avons indiqué, nous considérons dans cette thèse qu'il existe un unique superviseur en charge du système, et il est donc nécessaire de construire le diagnostic global du système. L'approche développée dans cette thèse étend les travaux effectués dans l'équipe DREAM et présentés dans [PC05].

1.5 Diagnostic de systèmes reconfigurables

Nous avons présenté jusqu'ici le diagnostic de systèmes à topologie statique. La plupart des systèmes ont cependant une topologie dynamique, mais ce point n'a pas beaucoup été traité dans la littérature.

Nous présentons d'abord les systèmes reconfigurables et insistons sur l'importance de la reconfiguration dans le diagnostic. Ensuite, nous présentons les quelques modélisations proposées sur le sujet.

1.5.1 Définition des systèmes reconfigurables

Un *système reconfigurable* [CTJK97] est un système dont le modèle peut changer au cours du temps. Cette modification est appelée *reconfiguration*.

[4] Les auteurs ne fournissant pas de terme, nous choisissons celui-ci.

Définition 1.11 (Reconfiguration)**.**
Étant donné un modèle du système, une reconfiguration *du système est une opération effectuée sur le système rendant son modèle obsolète.*

Une reconfiguration est une opération qui modifie le système d'une manière qu'on n'imaginait pas avant de le modéliser, ou qu'on n'a pas considérée parce que la taille du modèle du système aurait été trop importante. À l'issue de la reconfiguration, le modèle du système est modifié. Dans cette thèse, nous considérons en particulier qu'une reconfiguration est l'ajout ou la suppression de composants ou de connexions entre les composants.

Considérons le cas d'un réseau de télécommunication. On dispose d'un modèle décrivant les communications entre les composants de ce réseau, les pannes qui peuvent survenir et comment celles-ci se transmettent d'un composant à un autre. Le responsable du réseau décide, pour améliorer les performances de son réseau, d'ajouter un composant supplémentaire et de modifier la topologie du réseau pour profiter pleinement de ce composant. Le nombre de manières différentes d'appliquer cette reconfiguration est quasiment illimité (surtout si on considère que le gestionnaire peut ajouter ou supprimer plusieurs composants). D'autre part, il est possible que le fonctionnement du nouveau composant ne soit pas connu au moment de la modélisation du système d'origine (notamment dans le cadre du diagnostic en-ligne, lorsqu'un nouveau type de composant est ajouté au système). Ainsi, le modèle du système ne peut pas prendre en compte toutes les possibilités d'évolution possibles du réseau. Il faut donc utiliser des techniques permettant de changer le modèle du système en cours de diagnostic.

1.5.2 Importance du diagnostic de systèmes reconfigurables

Les systèmes reconfigurables sont très nombreux et couvrent presque tous les exemples de systèmes décentralisés par nature. On peut de nouveau citer les réseaux (réseaux de télécommunication, réseaux électriques, réseaux des eaux, *etc.*), les véhicules automobiles, les chaînes de production en usine, les *web-services*, *etc.*

On peut citer de nombreux cas justifiant une reconfiguration :

- Une procédure (automatique ou dirigée par un opérateur humain) peut isoler une partie défectueuse du système pour protéger le reste du système.
- Certains contextes nécessitent de s'adapter à une requête sur le système. Par exemple, les besoins en électricité d'entreprises ou de villes varient de manière régulière dans une journée, une semaine ou une année. Le réseau doit s'adapter pour fournir une qualité de service acceptable.
- Dans certains contextes, les connexions entre composants évoluent extrêmement vite. Dans le cas des *web-services* par exemple, chaque service peut faire appel à un certain nombre d'autres services et créer une connexion entre eux deux.
- Enfin, dans le cadre du diagnostic, il est parfois nécessaire de modifier le système pour discriminer les différents diagnostics possibles. Ainsi, en cas de défaillance d'une lampe, son utilisateur change généralement les piles et teste la lampe pour vérifier qu'elle fonctionne toujours.

Dans certains cas, on ajoute ou supprime des composants au système. Ainsi, l'opérateur peut décider d'ajouter de nouveaux composants plus puissants, de supprimer des composants obsolètes (inefficaces, dangereux, coûteux à maintenir) ou de remplacer les composants anciens pour maintenir ou améliorer les performances de son système.

Dans tous ces cas, il apparaît que le diagnostic ne doit pas s'arrêter à cause d'une reconfiguration. Commencer un nouveau diagnostic à la suite d'une reconfiguration est peu pratique dans le meilleur des cas (on perd le contexte du diagnostic) et impossible dans certains autres (dans le cas où les connexions évoluent très vite ou si on reconfigure pour raffiner un diagnostic).

Un système réel devant être diagnostiqué en-ligne subit forcément des reconfigurations à un moment ou un autre. Aussi, le diagnostic doit prendre en compte les reconfigurations.

1.5.3 Modélisations proposées

On peut citer deux modélisations permettant de manipuler des systèmes reconfigurables. Ces deux modélisations se basent sur les réseaux de Petri.

1.5.3.1 Reconfiguration par grammaires de graphes

L'utilisation des grammaires de graphes pour le diagnostic de systèmes reconfigurables a été introduite dans [BHFJ05]. Cet article considère les transformations de graphes par règle élémentaire comme présenté dans la figure 1.16.

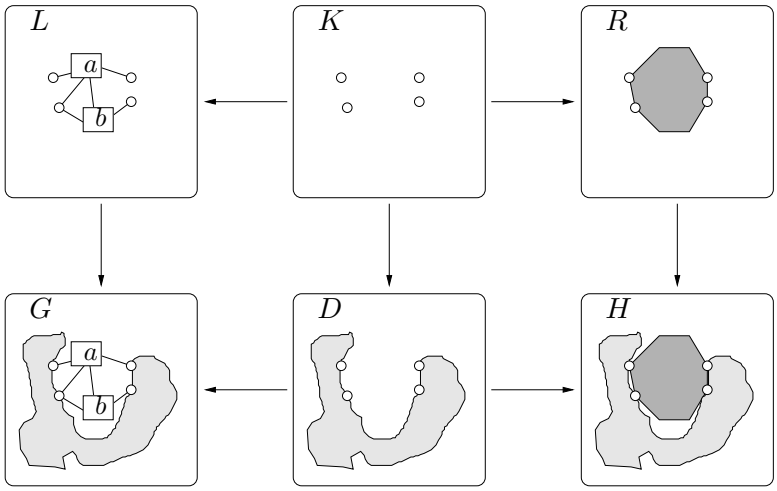

FIG. 1.16 – Les éléments d'une règle de transformation de graphe

La figure s'explique de la manière suivante : considérons un ensemble de nœuds (K). Formellement, il existe un morphisme injectif entre K et L, et un morphisme injectif entre K et R. Ensuite, on détecte une occurrence de L sur G (c'est-à-dire qu'il existe un morphisme injectif entre L et G). L'application de la transformation remplace la partie

du graphe L de G par R, produisant le graphe H. Il est ainsi possible de modifier les connexions entre composants et d'ajouter ou de supprimer des composants.

Dans [BHFJ05], les auteurs présentent une définition étendue des réseau de Petri. Sur ces réseaux de Petri, le franchissement de certaines transitions conduit à l'application d'une règle de transformation du graphe. Il existe donc un certain nombre de règles prédéfinies qui régissent les modifications qui peuvent être appliquées à la topologie du système.

1.5.3.2 Reconfiguration par hyper réseaux de Petri

Les hyper réseaux de Petri (*Petri Hypernets*) ont été introduits dans [BBPP04]. Dans un hyper réseau de Petri, certains jetons du réseau sont des réseaux de Petri. À leur tour, ces réseaux de Petri peuvent avoir pour jeton des réseaux de Petri. Considérons l'exemple de la figure 1.17. Les deux réseaux de Petri en haut de la figure correspondent au comportement d'un aéroport. Chaque réseau est un module manipulant un certain type de jetons. Le premier réseau est ainsi appelé un τ-module parce qu'il manipule des jetons de type *passager*. Au contraire, le deuxième réseau est un π-module, ce qui signifie qu'il manipule des jetons *avions*. Le comportement est le suivant : tout d'abord, un avion atterrit (on considère qu'il existe un niveau supérieur qui gère le déplacement des avions d'un aéroport à un autre). Les voyageurs débarquent. Puis l'avion fait le plein de carburant et se rend à la porte d'embarquement suivante. Les voyageurs embarquent et l'avion peut décoller.

Chaque jeton de type *avion* est représenté par un réseau de Petri (le réseau en bas de la figure). Chaque jeton de l'avion représente un voyageur ; le réseau de l'avion est donc d'un τ-module. Lorsqu'un jeton *avion* passe par la transition *embarquement* sur le réseau de l'aéroport, alors, un jeton *voyageur* emprunte de manière synchrone la transition *embarquement* sur le réseau de l'aéroport et ce jeton réapparaît dans le réseau de l'avion (ce phénomène est représenté par la flèche en pointillé). Ainsi, pendant que le jeton de type *avion* franchit la transition *embarquement*, il *capture* le jeton voyageur du premier réseau de l'aéroport.

Cette modélisation permet une représentation hiérarchique du système et une synchronisation des comportements d'un système. Les relations entre les différents réseaux de Petri sont représentées par le fait que certains réseaux de Petri sont des jetons dans d'autres réseaux. Puisqu'il est possible aux réseaux de Petri de passer d'un réseau à un autre, on peut assimiler ce comportement à un changement de topologie, et donc une reconfiguration.

1.5.4 Synthèse

Peu de travaux ont été effectués sur le diagnostic de systèmes reconfigurables. En particulier, les reconfigurations pour les modélisations par automate n'ont pas été considérées.

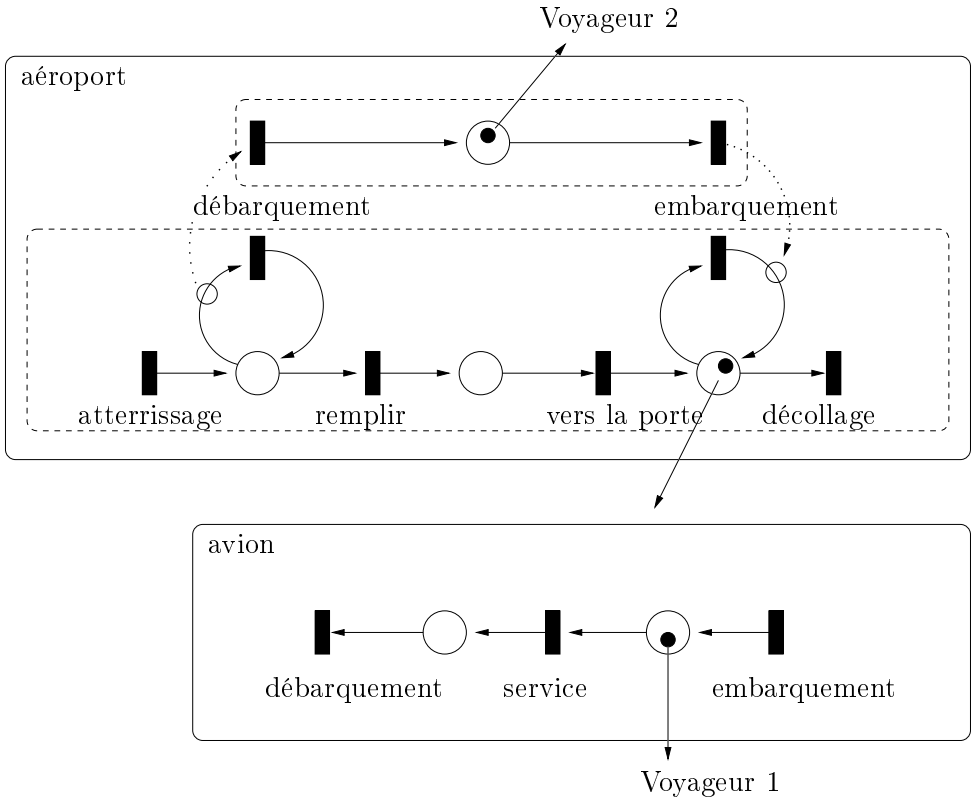

FIG. 1.17 – Exemple de synchronisation de transition : avion et aéroport

1.6 Résumé

Nous donnons ici un bref résumé des approches de la littérature que nous avons présentées dans ce chapitre. Les propriétés des différentes approches présentées sont résumées dans la table 1.1. Les notations utilisées dans le tableau sont les suivantes :

- MG : construction du modèle global nécessaire. Nous avons vu que le modèle global ne peut pas être construit pour les systèmes de taille réaliste.
- DDéc : diagnostic décentralisé.
- DDis. : diagnostic distribué.
- Conc. : approche représentant de manière efficace la concurrence. Ces techniques sont les techniques d'ordre partiel et l'utilisation de réseaux de Petri.
- CE : calcul en-ligne du diagnostic.
- OP : approche permettant de considérer un ordre partiel sur les observations.
- Inc. : approche permettant de considérer des incertitudes sur les observations.

Cette thèse se place dans la continuité des travaux effectués au sein de l'équipe DREAM, à savoir [TCJK96], [RC02] et surtout [PC05]. Notre approche s'appuie donc sur une modélisation sous forme d'automates. Puisqu'une modélisation globale du système

	MG	DDéc.	DDis.	Conc.	CE	OP	Inc.
[SSL+96]	Oui				Oui		
[WYL04]	Oui	Oui			Oui		
[RC02]	Oui				Oui		
[PC05]		Oui		Oui	Oui	Oui	
[LZ03b, LZ05]		Oui			Oui	Oui	Oui
[SW04]	Oui		Oui		Oui		
[CPR00]						Oui	Oui
[BFHJ03]			Oui	Oui	Oui	Oui	
[JB05]			Oui	Oui		Oui	

TAB. 1.1 – Résumé des articles présentés

est impossible, nous utilisons une modélisation décentralisée. Notre approche doit permettre de relâcher un certain nombre de contraintes sur les observations, en particulier permettre des incertitudes ou des pertes d'observations. Pour cela, nous utilisons une approche inspirée des techniques utilisées en [CPR00] et [LZ03b]. De plus, nous devons pouvoir diagnostiquer des systèmes reconfigurables.

Chapitre 2

Diagnostic de systèmes à événements discrets avec observations incertaines

Dans ce chapitre, nous considérons le diagnostic de systèmes à événements discrets. Nous définissons la notion de diagnostic tel que nous l'utilisons dans notre approche. Cette notion se fonde sur le modèle comportemental du système d'une part, et les observations émises par le système d'autre part, tous deux représentés par un automate. Ce chapitre considère uniquement le diagnostic *hors-ligne* et *global* de systèmes *non reconfigurables*. Le diagnostic se définit alors comme la synchronisation de l'automate du modèle et de l'automate des observations.

Dans une seconde partie, nous présentons les hypothèses considérées et relatives au transfert des observations du système vers le superviseur, ainsi que la méthode définie pour construire l'automate des observations dans ce contexte.

2.1 Définition des systèmes à événements discrets et modélisation

Nous considérons la définition de système à événements discrets telle que présentée à la définition 1.1 (page 7), à savoir *un système dynamique pouvant être modélisé par des variables prenant des valeurs dans un domaine discret et évoluant par l'occurrence d'événements discrets et instantanés*. Nous considérons des systèmes réactifs qui évoluent uniquement sur occurrence d'événements extérieurs, lesquels changent de manière synchrone l'état du système. Nous ne considérons pas l'aspect *reconfigurable* du système avant le chapitre 6.

Dans ce chapitre, nous ne considérons pas de modélisation décentralisée. Nous modélisons le système par un automate noté *Mod*.

Définition 2.1 (Modèle du système)**.**
Le modèle du système est un automate $Mod = (Q, E, T, I, F)$.

Nous reprenons la définition A.1 page 163, annexe A. Un automate est un tuple $Mod = (Q, E, T, I, F)$. Dans notre contexte, Q est l'ensemble fini d'états ; E l'ensemble fini d'événements qui peuvent avoir lieu sur le système ; T l'ensemble de transitions décrivant l'évolution du système lors de l'occurrence d'un ou plusieurs événements simultanés ; I est l'ensemble des états initiaux (au début de la période à diagnostiquer) ; et F est l'ensemble des états finaux (c'est-à-dire l'ensemble des états à l'issue du diagnostic).

Le modèle du système représente les comportements possibles du système sous forme de trajectoires définition A.3, page 164.

Définition 2.2 (Comportement)**.**
Un comportement *du système est représenté par une trajectoire sur le modèle Mod du système.*

Lorsqu'on décide de diagnostiquer un système, celui-ci peut se comporter *a priori* de toutes les manières possibles permises par le modèle. En particulier, l'état final peut être n'importe quel état. On considère donc que l'ensemble des états finaux pour un modèle est l'ensemble des états de l'automate : $F = Q$.

On distingue l'ensemble des *événements observables* noté $E_{Obs} \subseteq E$.

2.2 Définition des observations et modélisation

Dans cette thèse, nous considérons qu'il existe un opérateur unique appelé *superviseur* chargé de surveiller le comportement du système et devant intervenir sur celui-ci en cas de comportement anormal. Le superviseur reçoit des messages appelés *observations reçues* que nous décrivons par la suite.

Notre approche considère la représentation des observations sous la forme d'un automate. Nous présentons d'abord cette modélisation et nous illustrons ensuite l'intérêt d'une telle représentation.

2.2.1 Modélisation des observations

Nous reprenons les définitions que nous avons données dans la sous-section 1.2.3.2. Dans ce contexte, l'ensemble des événements pouvant se produire sur le système comprend un certain nombre d'*événements observables*. L'occurrence d'un événement observable génère une *observation émise* par le système. Cette observation peut être physiquement émise par un capteur, mais on considère que les observations sont émises par le système. L'observation émise peut être étiquetée par sa date d'émission par le capteur ou par le système, c'est-à-dire étiquetée par la date à laquelle l'événement observable a eu lieu. Il faut cependant garder à l'esprit que les capteurs ou les différents composants du système qui étiquettent ces messages ne disposent pas nécessairement de la même horloge, et les dates d'émission ne permettent généralement pas de retrouver l'ordre d'émission.

Les observations émises sont transmises au superviseur au moyen de canaux de communication. On considère que les éventuels capteurs font partie des canaux de communication. Les canaux de communication induisent un délai non négligeable et ont une fiabilité imparfaite qui peuvent induire une différence importante entre les observations émises et les observations *reçues* par le superviseur. Dans cette thèse, nous considérons les phénomènes suivants :

- Les canaux de communication peuvent fournir les observations reçues dans un ordre différent de leur émission. Ceci est dû aux délais qui peuvent varier d'un canal à un autre. Nous avons vu que dater l'émission des observations n'était pas suffisant pour retrouver l'ordre d'émission observations, à cause des différentes horloges des capteurs. Aussi, on dispose dans ce cas d'un ordre partiel d'observations.
- Dans certains cas, les observations reçues peuvent être incertaines. Dans ce cas, l'information contenue dans l'observation reçue indique qu'un événement observable parmi un ensemble donné s'est produit, sans qu'on sache lequel précisément. Parfois, on n'est même pas certain que l'événement ait réellement eu lieu.
- Il arrive que les canaux de communication perdent des observations. Dans ce cas de figure, on considère qu'on peut borner le nombre d'observations qui peuvent être perdues pendant un temps donné.
- Enfin, on considère la possibilité qu'un capteur tombe en panne et ne fournisse plus d'observation. On considère cependant qu'on sait lorsqu'un capteur tombe en panne.

Étant donnée une séquence d'observations reçues $\mathcal{O}$, il est possible de trouver un ensemble de séquences possibles d'observations émises. En effet, différentes séquences d'observations émises peuvent expliquer les observations reçues. Nous décidons de représenter ces observations sous la forme d'un automate.

Définition 2.3 (Automate des observations)**.**
*L'*automate des observations *est un automate* $Obs = (Q_o, E_o, T_o, I_o, F_o)$ *tel que* $E_{Obs} = E_o \cap E_m$ *où* E_m *est l'ensemble des événements du modèle du système.*

Nous donnons à cet automate le nom d'automate des observations, mais on devrait sans doute l'appeler en réalité l'*automate des événements observables*. En effet, les transitions de cet automate sont étiquetées non pas par des observations mais par des événements observables, et l'automate des observations représente donc les événements observables qui ont eu lieu sur le système. Par abus de langage, nous disons par la suite que cet automate représente les observations émises par le système.

L'automate des observations est construit à partir des observations reçues. On peut construire de différentes manières l'automate des observations. Il convient de donner une définition formelle pour représenter ce qu'est un automate des observations *correct*.

Définition 2.4 (Automate des observations correct)**.**
Un automate des observations Obs est correct *si l'ensemble des séquences d'événements étiquetant les trajectoires de cet automate est l'ensemble des séquences possibles d'événements observables compatibles avec le comportement des canaux de communication et les observations reçues.*

Cette définition indique qu'un automate des observations est considéré comme étant correct si chacune de ses trajectoires est étiquetée par des événements observables dont l'occurrence *explique* les observations reçues.

Les approches développées au sein de l'équipe DREAM considéraient jusqu'ici une séquence d'observations émises [Roz97], ou un ensemble partiellement ordonné [PC05]. L'utilisation d'un automate pour représenter les observations est très comparable à l'utilisation d'*index space* ([LZ04], présenté en 1.2.4.3, page 23) ou à l'approche utilisée pour représenter les observations dans [CPR00] (présenté en 1.2.4.1, page 19).

2.2.2 Illustration

Nous montrons dans cette sous-section l'intérêt d'utiliser un automate pour représenter les observations émises par le système. Comme nous l'avons vu, les observations reçues par le superviseur peuvent être très différentes des observations réellement émises. À partir d'une séquence d'observations reçues, il existe plusieurs séquences d'observations émises expliquant ces observations. Dans certains cas, le nombre de séquences est même infini (si on considère que des canaux de communication ont pu perdre des observations). Un automate permet de représenter de manière compacte ces différentes séquences possibles.

Nous donnons maintenant des exemples dans les différents cas que l'on considère au cours de cette thèse. Chaque exemple est très simple, mais il est possible de les combiner entre eux.

Observations complètement ordonnées Nous considérons que toutes les observations émises sont reçues, et toute observation reçue a été émise par le système. D'autre part, nous considérons que l'ordre d'émission des observations est connu. On a donc une seule séquence possible d'observations émises. Cette séquence peut être représentée par un automate dont les états sont en cascade.

Ainsi, prenons l'exemple de trois observations reçues correspondant aux événements observables : a, b et c (dans cet ordre). La figure 2.1 donne l'automate des observations pour cet exemple.

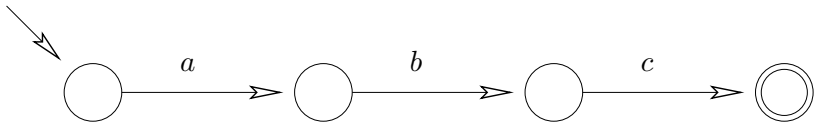

FIG. 2.1 – Automate des observations pour un ordre complet des observations

Ordre partiel des observations Nous considérons à présent que toutes les observations émises sont reçues, et toute observation reçue a été émise par le système. Cependant, il existe un ordre partiel entre les observations, c'est-à-dire que pour certains

couples d'observations, on ne sait pas quelle observation a été émise avant l'autre. Ce phénomène peut se représenter par différentes trajectoires sur l'automate.

Reprenons l'exemple précédent où l'on sait que l'observation correspondant à l'événement b a été émise avant celle correspondant à c, mais l'ordre entre les autres événements est inconnu. La figure 2.2 donne l'automate des observations pour cet exemple.

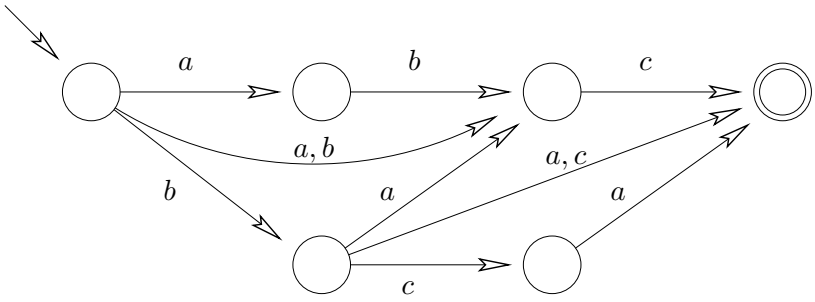

FIG. 2.2 – Automate des observations pour un ordre partiel des observations

Observations incertaines Nous considérons dans cet exemple que certaines observations sont incertaines, c'est-à-dire que plusieurs événements observables peuvent expliquer une observation. C'est par exemple le cas lorsqu'on dispose d'un capteur peu précis. Il est possible de représenter ce comportement par plusieurs transitions partant toutes du même état et menant toutes au même état, mais étiquetées par des événements observables différents. Si on considère que l'observation reçue a pu ne pas être émise, on peut étiqueter l'une des transitions par $\emptyset$ et déterminiser l'automate.

Considérons ainsi que l'on reçoive les observations des événements a, b et c, émises dans cet ordre. Considérons que l'événement b fait en réalité référence à l'événement observable b_1 ou l'événement observable b_2. La figure 2.3 donne l'automate des observations pour cet exemple.

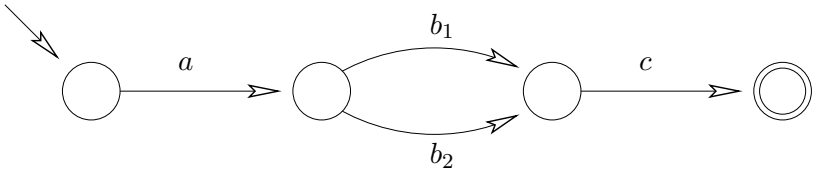

FIG. 2.3 – Automate des observations pour des observations incertaines

Perte d'observation À présent, nous considérons le cas où une ou plusieurs observations ont pu être perdues. Dans ce cas, on peut construire l'automate, puis le *cloner*. Chaque état du clone est relié à l'état de l'automate par des transitions étiquetées par les événements observables dont l'observation peut avoir été perdue. Chaque état du

clone représente en plus de la sémantique de l'état d'origine, le fait qu'une observation a été perdue.

Nous reprenons l'exemple que nous avons pris pour les observations complètement ordonnées. Nous considérons qu'au plus une observation a pu être perdue. Considérons que dans cet exemple, les seuls événements observables du système sont a, b et c. La figure 2.4 donne l'automate des observations pour cet exemple.

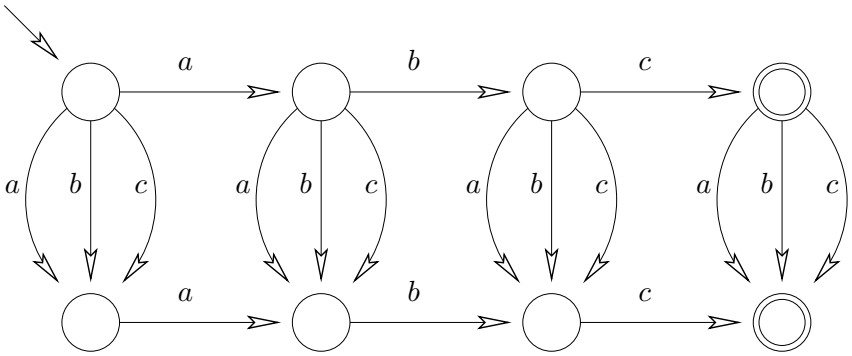

FIG. 2.4 – Automate des observations autorisant une observation perdue

Panne d'un capteur Nous considérons qu'un capteur peut tomber en panne, et qu'on est capable de déterminer que le capteur est tombé en panne. Dans ce cas, il est possible de représenter la panne du capteur par une boucle sur les états de l'automate, la boucle étant étiquetée par les événements observés par le capteurs.

Nous prenons un exemple où le superviseur a déterminé que le système a émis deux observations correspondant aux événements observables a et c dans cet ordre, et a déterminé qu'entre l'émission de ces deux observations, le capteur observant les événements b est tombé en panne. La figure 2.5 donne l'automate des observations pour cet exemple.

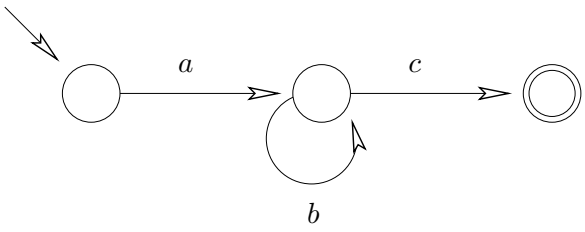

FIG. 2.5 – Automate des observations représentant un capteur en panne

Utiliser un automate des observations permet donc de relâcher un cerain nombre d'hypothèses effectuées généralement sur les canaux de communication.

2.3 Diagnostic

Dans cette section, nous donnons une définition formelle du diagnostic, que nous illustrons sur un exemple.

Le diagnostic consiste à *détecter, localiser et identifier tout dysfonctionnement dans le système*. Pour cela, le diagnostic doit repérer l'occurrence de certains événements appelés *événements de panne*. Nous avons vu dans le premier chapitre que le diagnostic de systèmes à événements discrets consistait à trouver la liste des comportements du système qui expliquent les observations (voir section 1.2.3, page 15). M. Sampath et coll. [SSL+95] définissent le diagnostic comme *l'ensemble des trajectoires du modèle qui expliquent la séquence d'observations émises*. Puisque la séquence d'observations émises est représentée par un automate, il est possible de représenter ces comportements par un automate et de calculer le diagnostic par la formule suivante :

Résultat 2.1.
Le diagnostic Δ hors-ligne et centralisé d'un système à événements discrets modélisé par Mod et ayant émis les observations représentées par Obs est calculé par :
$\Delta = Mod \otimes Obs$.

Exemple : Prenons l'exemple d'un système dont le comportement peut être modélisé par la figure 2.6. Nous avons représenté les états par des cercles simples, mais il faut garder à l'esprit que tous les états sont finaux. Les événements a, b et c sont observables et les événements w, x, y et z sont non observables.

Les observations sont représentées par l'automate de la figure 2.7.

Le diagnostic étant défini comme la synchronisation du modèle et de l'automate des observations, on obtient le résultat de la figure 2.8. L'automate n'a pas été complètement simplifié pour montrer qu'on n'atteint pas d'état final en suivant la transition étiquetée par $\{y, a\}$. Le diagnostic nous permet de connaître le comportement du système : $\{z\}$ suivi de $\{x, a, b\}$ et $\{w, c\}$, et éventuellement $\{z\}$. L'état final du système est soit 1 soit 2.

Remarquons que l'approche que nous avons proposée est très similaire à celle qui est utilisée dans le cadre de la vérification (*model-checking*) pour s'assurer des propriétés de sureté. On peut ainsi se référer à [Var95]. Nous donnons ici un résumé des résultats présenté dans [BBF+01b] (page 43 et suivantes) ou en français [BBF+01a] (pages 42 et suivantes).

On dispose d'un modèle du système sous la forme d'un automate $\mathcal{A}$. On souhaite s'assurer que le système respecte toujours une propriété notée ϕ exprimée en logique temporelle PLTL : $\mathcal{A} \models \phi$? Il est possible de construire un automate noté $\mathcal{B}_{\neg\phi}$ reconnaissant exactement toutes les exécutions qui ne satisfont pas ϕ. Alors, le problème de vérification se ramène au problème suivant : « Le langage reconnu par $\mathcal{A} \otimes \mathcal{B}_{\neg\phi}$ est-il vide ? » Si ce langage n'est pas vide, alors il existe au moins une exécution de $\mathcal{A}$ qui ne respecte pas ϕ, et la propriété n'est pas vérifiée.

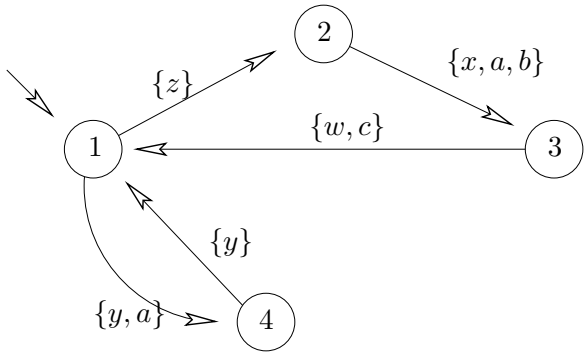

FIG. 2.6 – Exemple de modèle

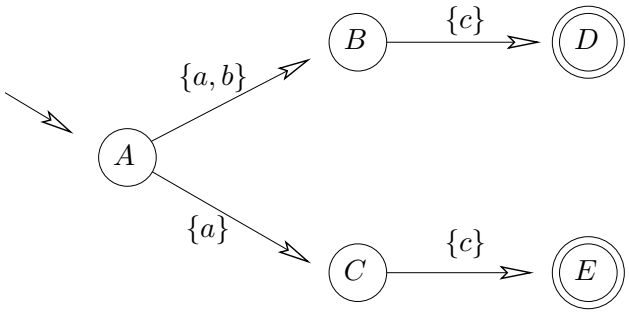

FIG. 2.7 – Exemple d'automate des observations

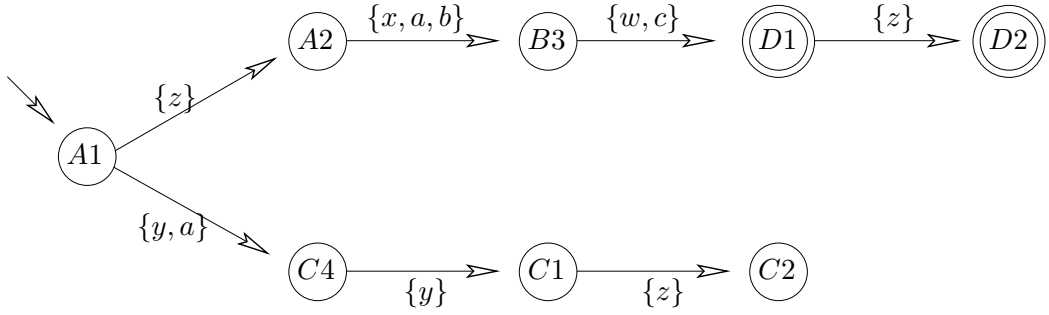

FIG. 2.8 – Diagnostic du modèle de la figure 2.6 avec les observations de la figure 2.7

Dans le cadre du diagnostic, on synchronise l'automate du modèle avec un automate des observations. Pour la vérification, on synchronise l'automate du modèle avec un automate représentant le comportement qu'on souhaite ne pas accepter. On voit que la correspondance est immédiate.

2.4 Construction de l'automate des observations

Nous donnons des exemples montrant comment construire automatiquement l'automate des observations. Ces exemples se basent sur certaines hypothèses effectuées sur le comportement des canaux de communication entre le système et le superviseur. Il est également possible de construire un automate pour d'autres hypothèses. Nous donnons tout d'abord ces hypothèses et présentons ensuite deux manières de construire l'automate des observations.

2.4.1 Hypothèses

Les hypothèses que nous considérons dans cette thèse reprennent les hypothèses utilisées dans [Pen02]. Dans ce contexte, les observations émises correspondent à l'émission d'un *message* (appelé aussi *alerte*) de la part du système au superviseur. Puisque nous représentons directement les événements observables grâce à l'automate des observations, nous n'avons pas besoin de représenter les observations émises.

Les canaux de communication ne perdent aucune observation et ne génèrent aucune observation. Les observations sont précises, mais il n'est pas possible de connaître l'ordre exact d'émission des observations, comme indiqué dans l'hypothèse suivante :

Hypothèse 2.1.
Les observations reçues sont sûres, sans perte et partiellement ordonnées.

Les observations reçues sont représentées par un couple comprenant le message émis et un identifiant unique.

Définition 2.5 (Modélisation d'une observation reçue)**.**
Une observation reçue *est modélisée par un couple* (e, id) *où* e *est l'événement observable ayant eu lieu et* id *un identifiant unique.*

Les observations ne sont pas nécessairement reçues dans l'ordre de leur émission. Cependant, pour certains couples observations (o_1, o_2), on est certain que l'observation o_1 a été émise avant l'observation o_2. On note $o_1 \prec o_2$. On considère aussi l'hypothèse suivante :

Hypothèse 2.2.
Deux observations du même événement observable sont ordonnées.

Soit deux observations $o = (e, id_1)$ et $o' = (e', id_2)$ avec $e = e'$. Alors, on a $(o \prec o') \vee (o' \prec o)$. Cette hypothèse permet de simplifier fortement les définitions.

2.4.2 Premier automate

Nous donnons à présent une première manière de construire l'automate des observations. Dans cette version, chaque état de l'automate indique une séquence d'ensembles d'observations émises par le système.

Définition 2.6 (Séquence possible)**.**
Soit $\mathcal{O}$ l'ensemble des observations reçues et $\prec$ la relation d'ordre partiel d'émission des observations. On dit que la séquence d'ensembles d'observations $(s_1, \ldots, s_m)$ est possible *si : $\forall i$, $\forall o \in s_i$, $\forall o' \in \mathcal{O}$, $o' \prec o \Rightarrow ((\exists j < i,\ o' \in s_j) \wedge (\nexists j,\ i \neq j \wedge o \in s_j))$.*

Nous donnons un exemple pour illustrer cette définition.

Exemple : Nous considérons l'exemple où nous avons reçu quatre observations $A = (a, 1)$, $B = (b, 2)$, $C = (c, 3)$ et $D = (a, 4)$. Nous considérons qu'étant données les hypothèses sur les canaux de communication, nous savons que $A \prec C$ et $A \prec D$. Cela signifie que nous savons que le message a de A a été émis avant le message c et avant le second message a. Dans ce contexte, $(\{A, B\}, \{D\})$ est une séquence possible. Cela signifie qu'il est possible que A et B aient été émis en même temps, puis D (C ayant été émis par la suite). En revanche, $(\{D\})$ n'est pas une séquence possible puisque A aurait dû être émis auparavant.

Résultat 2.2 (Premier automate des observations)**.**
Soit $\mathcal{O}$ l'ensemble des observations reçues et $\prec$ la relation d'ordre partiel d'émission des observations. L'automate des observations $Obs = (Q, E, T, I, F)$ défini par :

- *$Q = \{q \mid q$ est une séquence possible$\}$,*
- *$E = E_{Obs}$,*
- *$T = \{(q, l, q') \mid \exists s,\ q = (s_1, \ldots, s_m) \wedge q' = (s_1, \ldots, s_m, s) \wedge l$ est l'ensemble des événements observables de $s\}$,*
- *$I = \{()\}$ et*
- *F est l'ensemble des séquences complètes et possibles,*

est un automate correct.

Démonstration. Montrons que pour toute séquence d'observations *seq* complète et possible, il existe une trajectoire représentant cette séquence. Puisque cette séquence existe et puisqu'elle contient toutes les observations, $seq \in F$.

Montrons par récurrence que pour tout $q \in Q$, il existe un chemin depuis l'état initial vers q étiqueté par les observations de la séquence q. Soit $q = ()$, alors ce chemin (vide) existe ($q \in I$). Soit i tel que la propriété est vraie pour toute séquence de taille i. Soit $q_{i+1} = (e_1, \ldots, e_i, e_{i+1})$ une séquence de longueur $i + 1$. Alors, la propriété est vraie pour $q_i = (e_1, \ldots, e_i)$ et il existe une transition entre q_i et q_{i+1} étiquetée par les événements observables de e_{i+1}. Donc, la propriété est vraie pour q_{i+1}, et donc pour tout $q \in Q$.

Puisqu'il existe un chemin de $q_0 \in I$ à $seq \in F$ étiqueté par les événements observables, alors il existe une trajectoire.

En outre, tout état $q \in F$ est associé à une trajectoire, et il n'existe qu'une transition entrante par état. Ainsi, la seule trajectoire menant à $q \in F$ est la trajectoire correspondant à l'émission des observations de q. Donc, il n'y a pas de trajectoire supplémentaire.

Cet automate des observations est donc correct. □

Exemple : La figure 2.9 donne l'automate des observations construit pour l'exemple présenté précédemment avec la méthode du résultat 2.2. Les séquences associées à chaque état ne sont pas représentées. Sur cet exemple, on voit que le nombre d'états est très important alors que l'exemple ne comprend que 4 observations. La taille de l'automate augmente de manière exponentielle avec le nombre des observations. On remarque notamment qu'il serait possible de fusionner tous les états tels que les observations dans la séquence sont identiques. Ainsi, dans la figure 2.9, tous les états grisés correspondent à l'émission des trois observations A, B et C : respectivement $(\{A\},\{B\},\{C\})$, $(\{A\},\{B,C\})$, $(\{A\},\{C\},\{B\})$, $(\{A,B\},\{C\})$ et $(\{B\},\{A\},\{C\})$. Tous ces états n'ont qu'une seule transition sortante étiquetée par a (l'événement D) menant à un état final. Ces états sont donc équivalents. C'est pourquoi nous proposons une autre manière de construire l'automate.

2.4.3 Automate par ensembles d'observations

On remplace à présent les séquences possibles par des ensembles possibles.

Définition 2.7 (Ensemble possible)**.**
Soit $\mathcal{O}$ l'ensemble des observations reçues et $\prec$ la relation d'ordre partiel d'émission des observations. On dit que l'ensemble $O \subseteq \mathcal{O}$ est un ensemble possible*, noté $\lhd(O)$, si $\forall o \in O, \forall o' \in \mathcal{O}$, $o' \prec o \Rightarrow o' \in O$.*

Cette définition indique qu'un ensemble O est possible s'il contient toutes les observations qui précèdent une des observations de O. Remarquons que si $seq = (s_1, \ldots, s_i)$ est une séquence possible, alors $s_1 \cup \cdots \cup s_i$ est un ensemble possible. Et réciproquement, si O est un ensemble possible, il existe une partition de O qui est permet de former une séquence possible.

Par la suite, le terme *possible* fait référence à un *ensemble possible*.

Exemple : Nous reprenons l'exemple précédent. La liste des possibles est : $\emptyset$, $\{A\}$, $\{B\}$, $\{A,B\}$, $\{A,C\}$, $\{A,D\}$, $\{A,B,D\}$, $\{A,B,C\}$, $\{A,C,D\}$ et $\mathcal{O}$. Considérons par exemple $O = \{A,B,D\}$. Cet ensemble d'observations est un possible ; cela signifie qu'il est possible que, à une date donnée t, O ait été l'ensemble des observations émises. En revanche, $\{D\}$ seul n'est pas un possible, puisque l'observation A est préalable à D.

L'automate des observations peut être défini en utilisant les ensembles possibles. Dans l'automate que nous construisons, un état représente l'ensemble des observations émises à une date donnée. Les états de l'automate sont donc des ensembles d'observations ($q \subseteq \mathcal{O}$). Un état q doit être un ensemble possible pour être atteignable.

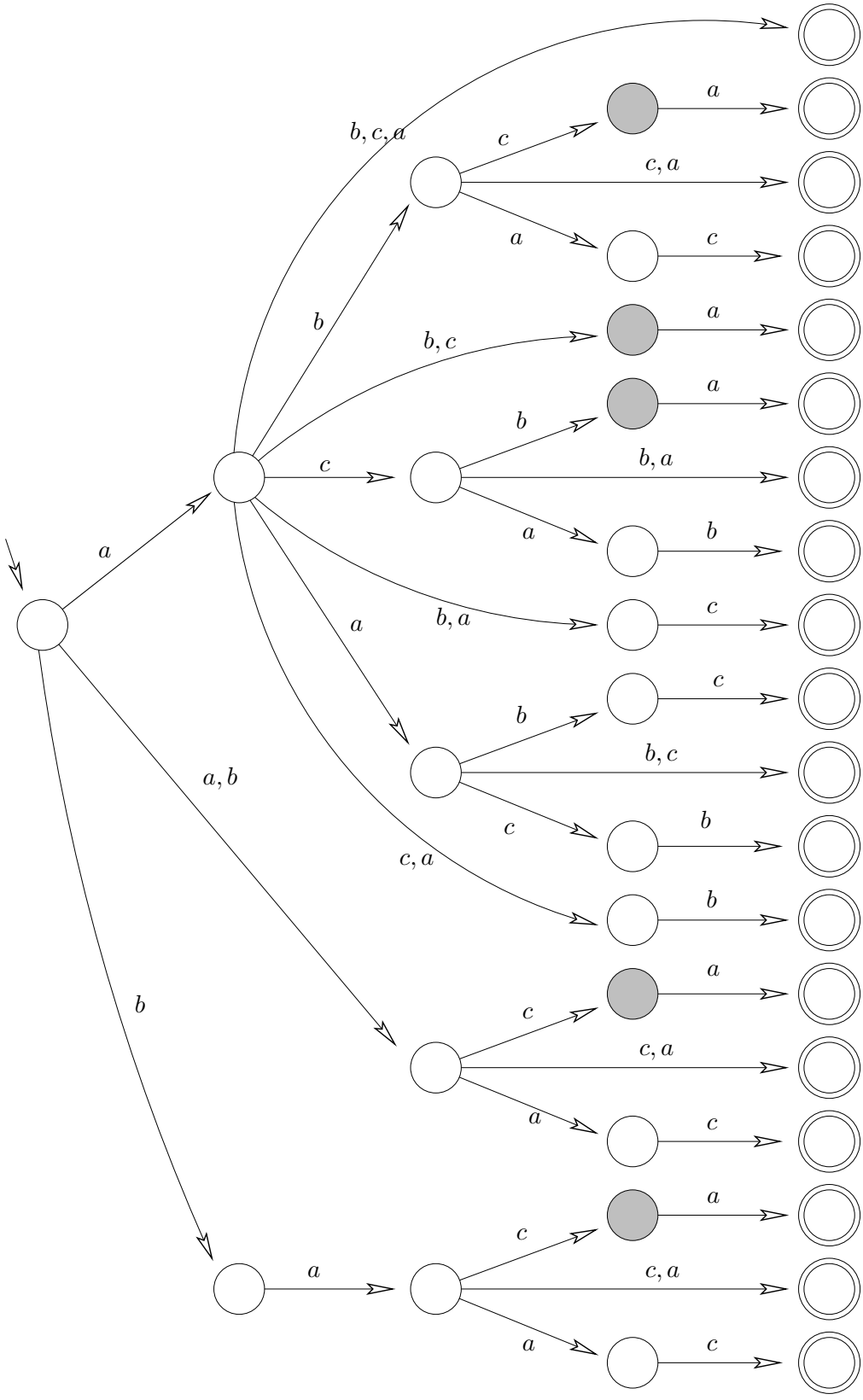

FIG. 2.9 – Exemple d'automate des observations construit à l'aide du résultat 2.2

Résultat 2.3 (Automate des observations)**.**
Soit $\mathcal{O}$ l'ensemble des observations reçues et $\prec$ la relation d'ordre partiel d'émission des observations. L'automate des observations $Obs = (Q, E, T, I, F)$ défini par :

- $Q = \{q \subseteq \mathcal{O} \mid \lhd (q)\}$,
- $E = E_{Obs}$,
- $T = \{t = (q, l, q') \mid \exists q'' \subseteq \mathcal{O}, q' = q \uplus q'' \wedge (\nexists o, o' \in q'',\ o \prec o')\ \wedge l$ *est l'ensemble des messages de* $q''\}$,
- $I = \{\emptyset\}$, *et*
- $F = \{\mathcal{O}\}$,

est un automate correct.

Démonstration. Nous montrons que le résultat 2.3 est bien un automate des observations correct. Pour cela, nous montrons la condition de la définition 2.4.

Soit une séquence $(s_1, \ldots, s_m)$ d'observations reçues. Alors, on a $s_1 \cup \cdots \cup s_m = \mathcal{O}$ (puisque toutes les observations émises ont été reçues et puisqu'il n'y a pas de perte). Soit $q_i = s_1 \cup \cdots \cup s_i$ et $e_i = \{e \mid \exists id, o,\ o \in s_i \wedge o = (e, id)\}$. Alors, $((q_0, \ldots, q_m), (e_1, \ldots, e_m))$ est une trajectoire de Obs.

Soit une trajectoire $((q_0, \ldots, q_m), (e_1, \ldots, e_m))$ de Obs. Alors, la séquence $(q_1, q_2 \setminus q_1, \ldots, q_m \setminus q_{m-1})$ correspond à cette trajectoire. Or, cette séquence est une séquence possible des observations émises. □

Les états sont des ensembles possibles d'observations. L'unique état initial correspond à l'ensemble vide (aucune observation n'a encore été émise) et l'unique état final correspond à l'ensemble des observations émises $\mathcal{O}$. Une transition de q à q' décrit l'émission des observations q''. Deux observations o et o' ne peuvent être émises en même temps si $o \prec o'$. L'étiquette de la transition est l'ensemble des événements e correspondant aux observations q''.

Exemple : L'automate des observations est présenté à la figure 2.10. On voit par exemple que l'état AC représente le fait que le système ait envoyé les observations A et C. À partir de cet état, la transition étiquetée par a, b représente l'émission simultanée des observations B et D. À l'issue de cette émission, l'état du système est $ABCD$, c'est-à-dire que l'ensemble des observations $\mathcal{O}$ a été émis. L'état grisé de cette figure correspond aux cinq états grisés de la figure 2.9.

2.5 Conclusion

Dans ce chapitre, nous avons proposé de représenter les observations émises par le système sous forme d'un automate appelé *automate des observations*. Les hypothèses sur les canaux de communication permettent de construire l'automate des observations à partir de la séquence d'observations reçues.

Le diagnostic de systèmes à événements discrets peut se traduire par la synchronisation de deux automates : l'automate *Mod* représentant le comportement du système et l'automate *Obs* représentant les observations émises par le système.

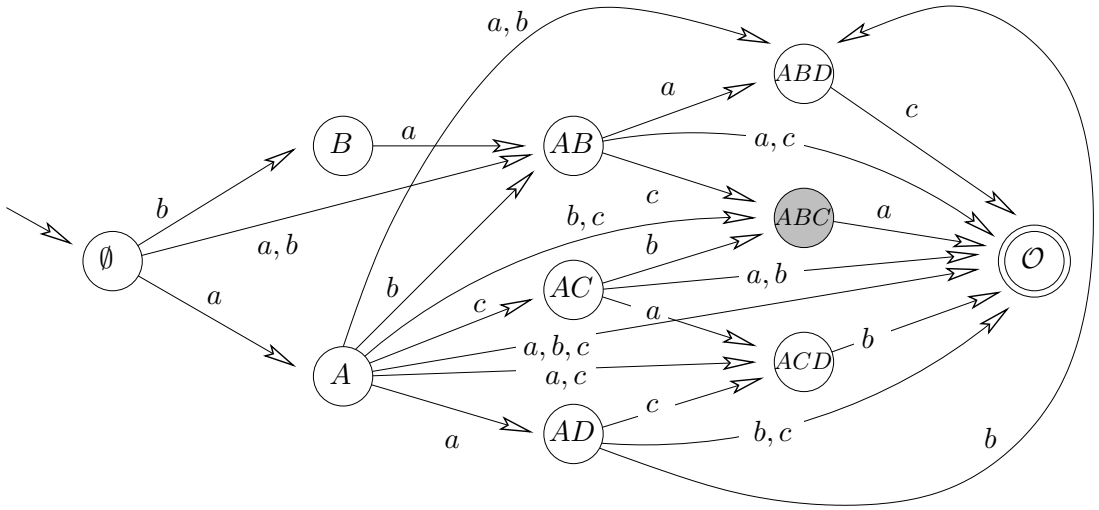

FIG. 2.10 – Exemple d'automate des observations construit à l'aide du résultat 2.3

Ce résultat permet d'avoir une définition formelle du diagnostic. Il est à présent possible de développer des outils mathématiques pour améliorer les performances des algorithmes de diagnostic et permettre le diagnostic incrémental et en-ligne.

Nous avons également montré deux méthodes pour construire l'automate des observations dans le cadre des systèmes que nous considérons généralement. La seconde méthode est plus économe en terme d'états de l'automate, mais nous avons gardé la première parce qu'elle nous est utile par la suite.

Chapitre 3

Calcul incrémental du diagnostic hors-ligne par chaînes d'automates

Nous souhaitons maintenant nous intéresser au diagnostic en-ligne. Nous avons vu en section 1.3 que pour être capable d'effectuer un diagnostic en-ligne, il était nécessaire de savoir faire un calcul incrémental du diagnostic. La méthode que nous proposons dans ce chapitre consiste à construire un diagnostic pour une fenêtre d'observations et d'utiliser ce diagnostic efficacement pour le calcul d'une fenêtre plus grande. Ce calcul est considéré hors-ligne et l'extension au calcul en-ligne est étudiée dans le chapitre suivant.

Pour cela, nous définissons une structure appelée *chaîne d'automates* qui permet de manipuler des automates *découpés par morceaux*. Le diagnostic est calculé à partir d'un découpage par morceaux de l'automate des observations, et il est possible de *reconstruire* le diagnostic de la fenêtre globale. Dans ce cadre du diagnostic incrémental, on considère qu'on ajoute un morceau à l'automate des observations. Alors, il est suffisant d'effectuer un diagnostic pour ce morceau et d'ajouter le morceau de diagnostic obtenu au diagnostic précédemment construit.

Ce chapitre se divise de la manière suivante : dans un premier temps, nous donnons une définition formelle du calcul incrémental du diagnostic. Puis, nous présentons les chaînes d'automate et montrons comment il est possible, en se basant sur un découpage de l'automate des observations, de calculer (incrémentalement ou non) le diagnostic. Une opération appelée *diagnostic incrémental* particulièrement adaptée au calcul incrémental du diagnostic est expliquée en section 3.3. L'opération de découpage de l'automate des observations, nécessaire à un calcul incrémental du diagnostic, est explicitée en section 3.4.

3.1 Définition du calcul incrémental

Nous reprenons ici les définitions que nous avons fournies au paragraphe 1.3.1.2, page 28. Dans ce cadre, une *fenêtre* $\mathcal{W}$ est une partie des observations émises par le système. Cette fenêtre est la plupart du temps *temporelle*, c'est-à-dire qu'elle considère

une période donnée, mais ce n'est pas le cas de manière générale. On peut effectuer un diagnostic $\Delta_{\mathcal{W}}$ sur la fenêtre $\mathcal{W}$. Ce diagnostic explique alors les observations de la fenêtre.

Nous reprenons la définition de *calcul incrémental du diagnostic* que nous avons donnée à la définition 1.10, page 28 : *étant donnée une fenêtre $\mathcal{W}$ et un diagnostic $\Delta_{\mathcal{W}}$, étant donnée une fenêtre $\mathcal{W}'$ dont $\mathcal{W}$ est un préfixe, le calcul incrémental du diagnostic de la fenêtre $\mathcal{W}'$ est le calcul du diagnostic de la fenêtre $\mathcal{W}'$ utilisant $\Delta_{\mathcal{W}}$ et la fenêtre $\mathcal{W}' - \mathcal{W}$.*

Dans le cadre du diagnostic tel que nous l'avons présenté au chapitre 2, on peut définir le calcul incrémental du diagnostic de manière plus précise.

Une fenêtre est un automate des observations qui représente une partie des observations émises par le système. Nous devons d'abord définir formellement dans notre contexte le terme de préfixe que nous utilisons dans la formule de calcul incrémental. On dit qu'un automate *Obs* est préfixe de *Obs'* si pour toute trajectoire chem de *Obs'*, il existe un préfixe de chem dans l'automate *Obs*.

Définition 3.1 (Automate préfixe)**.**
Un automate A est dit préfixe *d'un autre automate A' si, pour toute trajectoire* $\text{chem}' = ((q_0, \ldots, q_n), (l_1, \ldots, l_n))$ *de A', il existe $i \in \{0, \ldots, n\}$ tel que $((q_0, \ldots, q_i), (l_1, \ldots, l_i))$ est une trajectoire de A.*

On remarque que l'on considère qu'on ajoute des observations à la fin de l'automate. On ne considère pas la possibilité d'ajouter des observations au début de l'automate, ni d'insérer des observations au milieu des autres observations (comme cela peut être le cas pour [LZ04]). Nous effectuons en effet l'hypothèse que tout automate des observations est correct pour une fenêtre des observations donnée, ce qui exclut qu'il soit possible d'insérer une observation au milieu d'une séquence d'observations préalablement utilisée.

Le calcul incrémental du diagnostic par automate peut donc se définir de la manière suivante :

Définition 3.2 (Calcul incrémental du diagnostic par automate)**.**
Soit Obs un automate des observations et $\Delta_{Obs} = Mod \otimes Obs$ le diagnostic utilisant cet automate des observations. Soit Obs' un automate des observations tel que Obs est un préfixe de Obs'. Alors, le calcul incrémental du diagnostic par automate *consiste à utiliser Δ_{Obs} et l'extrait $Obs' - Obs$ pour calculer $\Delta_{Obs'} = Mod \otimes Obs'$.*

L'extrait $Obs' - Obs$ évoqué dans la définition 3.2 est un morceau d'automate. Il est explicité dans la section suivante. Dans le contexte incrémental, la complexité du diagnostic de *Obs'* doit dépendre uniquement de la taille de l'extrait $Obs' - Obs$, mais pas de la taille de *Obs'* ni *Obs*.

3.2 Chaînes d'automates et application au calcul incrémental du diagnostic

Le formalisme de chaîne d'automates a été développé pour permettre le calcul incrémental du diagnostic.

Nous donnons tout d'abord les définitions de chaînes d'automates ainsi que les liens entre une chaîne d'automates et un automate. Ensuite, nous redéfinissons les opérateurs utilisés pour le diagnostic dans le cadre des chaînes d'automates. Puis, nous présentons une première application au diagnostic.

3.2.1 Chaînes d'automates et découpage correct

Dans un automate classique, on se déplace d'un état à un autre *via* les transitions. Les chaînes d'automates proposent de *découper* l'automate en *morceaux* d'automates plus petits pour permettre un raisonnement sur chacun des automates séparément.

Définition 3.3 (Chaîne d'automates)**.**
Une séquence d'automates $(A^1, \ldots, A^n)$ *avec* $A^i = (Q^i, E^i, T^i, I^i, F^i)$ *est appelée* chaîne d'automates *et notée* $\mathcal{E}_A$ *si :*

- $\forall i, j,\ E^i = E^j$,
- $\forall i, j,\ j > i, \forall q,\ q \in Q^i \cap Q^j \Rightarrow q \in F^i \wedge q \in I^{i+1}$, *et*
- $\forall i, j, \forall q, q',$ *si* $\{q, q'\} \subseteq Q^i \cap Q^j$ *alors* $\forall p$, *chemin de* A^i *entre* q *et* q', p *est aussi un chemin de* A^j.

Par la suite, le numéro i placé en exposant fait référence à l'automate numéro i de la chaîne d'automates. De plus, le terme de *morceau* fait référence à un automate d'une chaîne.

Dans [GCL05b], nous avons donné une première définition de chaînes d'automates dans laquelle seule la première condition était nécessaire. Par la suite, dans [GCL05c, GCL05a], nous avons donné une définition plus contrainte permettant d'obtenir des propriétés supplémentaires. Les trois conditions de la définition de chaîne d'automates sont justifiées plus loin (pages 68 et 70). Remarquons cependant les propriétés de la deuxième condition. Soit une chaîne d'automates comprenant 10 automates, et considérons que les automates 3 et 7 comportent le même état q. Alors, q est un état final du troisième automate de la chaîne et un état initial du quatrième. Puisqu'il s'agit d'un état des automates 4 et 7, on peut récursivement utiliser la deuxième condition pour conclure que les automates 5 et 6 comportent cet état. Ainsi, tous les automates entre 3 et 7 comportent l'état q et cet état est final pour tous les automates (sauf éventuellement le dernier) et initial pour tous les automates (sauf éventuellement le premier).

Un exemple de chaîne d'automates composée de trois automates est présenté sur la figure 3.1. Notons que si le deuxième A^2 de la chaîne ne comporte pas de transition étiquetée par c, cet événement fait cependant parti de l'ensemble des événements E^2 de A^2. Notons que les états communs (états 3, 4, 5 et 9) à deux automates sont finaux dans le premier et initiaux dans le second. On remarque aussi que les chemins entre 4

et 5 présents dans l'automate A^2 sont présents dans l'automate A^3, et réciproquement comme requis par la troisième condition de la définition.

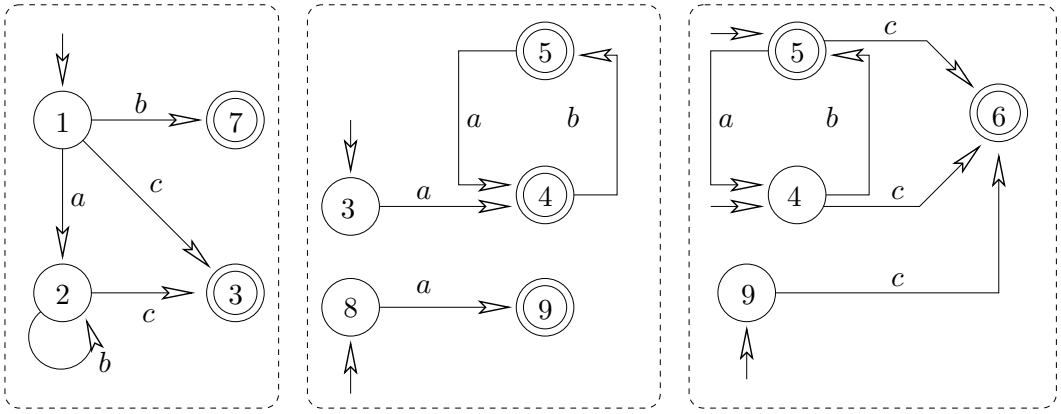

FIG. 3.1 – Exemple de chaîne d'automates

Un automate représente un ensemble de trajectoires. Une chaîne d'automates représente également un ensemble de trajectoires.

Définition 3.4 (Chemin de chaîne)**.**
Soit $\mathcal{E}_A = (A^1, \ldots, A^n)$ *une chaîne d'automates. Soient* $i \in \{1, \ldots, n\}$ *et* $j \in \{i, \ldots, n\}$*, et soient* $(j - i + 1)$ *chemins* $\text{chem}^i, \ldots, \text{chem}^j$ *tels que* $\forall k \in \{i, \ldots, j\}$ *:*
- $\text{chem}^k = ((q_0^k, \ldots, q_{mk}^k), (l_1^k, \ldots, l_{mk}^k))$ *est un chemin sur* A^k*,*
- $k \neq i \Rightarrow q_0^k \in I^k$*,*
- $k \neq j \Rightarrow q_{mk}^k \in F^k$*,*
- $k \neq j \Rightarrow q_0^{k+1} = q_{mk}^k$*.*

Alors, $\text{chem} = ((q_0^i, \ldots, q_{mi}^i, q_1^{i+1}, \ldots, q_{mj}^j), (l_1^i, \ldots, l_{mi}^i, l_1^{i+1}, \ldots, l_{mj}^j))$ *est un* chemin *sur* $\mathcal{E}_A$*.*

Un chemin sur une chaîne d'automates est une succession de transitions partant d'un état q de l'automate A^i jusqu'à un état q_{mi}^i de F^i suivi d'une succession de transitions depuis $q_0^{i+1} = q_{mi}^i$ sur l'automate A^{i+1}, et ainsi de suite jusqu'à un état de A^j. Ainsi, $\text{chem}_1 = ((2, 3, 4), (c, a))$ est un chemin de la chaîne d'automates présentée sur la figure 3.1. De même, $\text{chem}_2 = ((8, 9, 6), (a, c))$. Le passage d'un automate A^i de la chaîne au suivant A^{i+1} se fait grâce à un état $q \in F^i \cap I^{i+1}$ (par exemple, l'état 3 pour la passage de l'automate A^1 à A^2 sur la chaîne de la figure 3.1). Ces états sont appelés *états frontière.*

Définition 3.5 (Trajectoire de chaîne)**.**
Soit $\mathcal{E}_A = (A^1, \ldots, A^n)$ *une chaîne d'automates. Un chemin de* $\mathcal{E}_A$ $\text{chem} = ((q_0, \ldots, q_m), (l_1, \ldots, l_m))$ *est une* trajectoire *de* $\mathcal{E}_A$ *si* $q_0 \in I^1$ *et* $q_m \in F^n$*.*

Sur l'exemple de la figure 3.1, $\text{chem} = ((1, 3, 4, 5), (c, a, b))$ est une trajectoire de la chaîne d'automates. On dit que chem est la *reconstruction* des 3 trajectoires $\text{chem}^1 =$

$((1,3),(c))$, $\text{chem}^2 = ((3,4,5),(a,b))$ et $\text{chem}^3 = ((5),())$. Remarquons qu'il est possible de trouver d'autres trajectoires telles que leur reconstruction donne chem. En revanche, il est impossible de trouver une trajectoire passant par les états 7, 8 ou 9. Ces états sont donc *inutiles* et nous voyons en section 3.3 une manière de les ôter. Une chaîne d'automates permet donc de représenter des trajectoires.

Théorème 3.1.
Soit $\mathcal{E}_A = (A^1, \ldots, A^n)$ une chaîne d'automates. Soit $k \in \{1, \ldots, n\}$ et soit la séquence d'automates $\mathcal{E}'_A = (A'^1, \ldots, A'^n)$ telle que A'^k est la simplification de A^k et $\forall j \neq k,\ A'^j = A^j$. Alors $\mathcal{E}'_A$ est une chaîne d'automates. De plus, les ensembles des trajectoires de $\mathcal{E}_A$ et de $\mathcal{E}'_A$ sont identiques.

Nous rappelons que la *simplification* est l'opération définie en fin de section A.1 qui consiste à supprimer les transitions et états dans un automate qui n'apparaissent dans aucune trajectoire.

Démonstration. On considère la notation utilisée dans le théorème. Il nous faut tout d'abord prouver les trois conditions de la définition de chaîne d'automates :

- $\forall i, j,\ E'^i = E'^j$. Cette condition est vérifiée.
- $\forall i, j,\ j > i, \forall q,\ q \in Q'^i \cap Q'^j \Rightarrow q \in F'^i \wedge q \in I'^{i+1}$. Si $i \neq k$ et $j \neq k$, alors la propriété est vérifiée puisqu'elle l'était pour la chaîne d'automate $\mathcal{E}_A$. Si $i = k$, alors $F'^i = F^i \cap Q'^i$. Donc, si $q \in Q'^i \cap Q'^j$, alors $q \in Q^i \cap Q^j$ et donc $q \in F^i \cap I^{i+1}$ (puisque $\mathcal{E}_A$ est une chaîne d'automates). Ainsi, $q \in (F^i \cap Q'^i) = F'^i$ et $q \in I^{i+1} = I'^{i+1}$. On peut effectuer un raisonnement équivalent pour $j = k$. Donc, la condition est vérifiée pour A'^i et A'^j.
- $\forall i, j, \forall q, q'$, si $\{q, q'\} \subseteq Q'^i \cap Q'^j$ alors $\forall$chem, chemin de A'^i entre q et q', chem est aussi un chemin sur A'^j. Cette condition est vérifiée pour $i \neq k \wedge j \neq k$. Pour $i = k$, le chemin chem est un chemin de A'^i, donc de A^i. Ainsi, c'est un chemin de A^j et donc de $A'^j = A^j$. Enfin, pour $j = k$, si les états q et q' appartiennent à I'^j, alors la simplification n'a pas supprimé les transitions du chemin chem. Donc, la condition est respectée.

$\mathcal{E}'_A$ est donc une chaîne d'automates.

Toute trajectoire de $\mathcal{E}'_A$ est bien sûr une trajectoire de $\mathcal{E}_A$. Toute trajectoire de $\mathcal{E}_A$ est la reconstruction de la séquence de trajectoires $(\text{chem}^1, \ldots, \text{chem}^n)$. Chaque trajectoire chem^j est une trajectoire de A^j et donc de A'^j. Ainsi, chem est une trajectoire de $\mathcal{E}'_A$. □

Grâce à ce théorème, il est possible de simplifier les automates de la chaîne sans perdre de trajectoire.

Le but d'un automate est de représenter des trajectoires, et nous avons vu que les chaînes d'automates ont également des trajectoires. Le but est maintenant d'utiliser des chaînes d'automates à la place de certains automates. Nous définissons donc une correspondance entre automates et chaînes d'automates.

Définition 3.6 (Découpage correct)**.**
On dit qu'une chaîne d'automates $\mathcal{E}_A$ est un découpage correct *de l'automate A ssi l'ensemble des événements E de A est le même que l'ensemble des événements de la*

chaîne ($E^1 = \cdots = E^n = E$) *et l'ensemble des trajectoires de* $\mathcal{E}_A$ *est identique à l'ensemble des trajectoires de* A.

Définition 3.7 (Reconstruction)**.**
On appelle reconstruction *de la chaîne d'automates* $\mathcal{E}_A = (A^1, \ldots, A^n)$ *où* $\forall i,\ A^i = (Q^i, E^i, T^i, I^i, F^i)$ *l'automate* $A = (Q, E, T, I, F)$ *défini comme suit :*

- $Q = Q^1 \cup \cdots \cup Q^n$,
- $E = E^1 = \cdots = E^n$,
- $T = T^1 \cup \cdots \cup T^n$,
- $I = I^1$ *et*
- $F = F^n$.

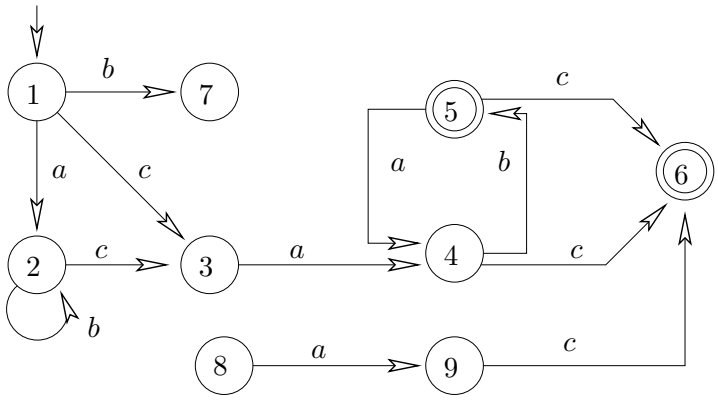

FIG. 3.2 – Reconstruction (non simplifiée) de la chaîne d'automates de la figure 3.1

Reconstruire la chaîne d'automates est une opération consistant à regrouper les n automates de la chaîne en un seul automate où les états initiaux sont les états initiaux du premier automate, et les états finaux ceux du dernier automate. La reconstruction (non simplifiée) de la chaîne présentée sur la figure 3.1 est donnée sur la figure 3.2.

On voit que la chaîne d'automates de la figure 3.1 est un découpage correct de l'automate de la figure 3.2. Nous démontrons ce résultat.

Théorème 3.2.
Soit $\mathcal{E}_A$ *une chaîne d'automates et* A *un automate. Si* $\mathcal{E}_A$ *est un découpage correct de* A*, alors* A *peut être obtenu par reconstruction de* $\mathcal{E}_A$.

Démonstration. Soit $A = (Q, E, T, I, F)$ un automate et $\mathcal{E}_A = (A^1, \ldots, A^n)$ une chaîne d'automates avec $A^i = (Q^i, E^i, T^i, I^i, F^i)$ tel que $\mathcal{E}_A$ est un découpage correct de A. Soit $A_R = (Q_R, E_R, T_R, I_R, F_R)$ une reconstruction de la chaîne $\mathcal{E}_A$. Il nous faut prouver que l'ensemble des trajectoires de A (qui est le même que celui-ci de $\mathcal{E}_A$) est le même que A_R.

Notons $\mathcal{E}_{A_{1,2}} = (A^1, A^2)$. $\mathcal{E}_{A_{1,2}}$ est une chaîne d'automates. Soit $A_{1,2}$ la reconstruction de $\mathcal{E}_{A_{1,2}}$. Nous donnons d'abord deux remarques sur cette chaîne.

Soit $t = (q, l, q')$, une transition de $A_{1,2}$.

1. $\{q, q'\} \subseteq Q^1$ ou $\{q, q'\} \subseteq Q^2$ (puisque $(q, l, q') \in T_{1,2} = T^1 \cup T^2$). Par conséquent, si l'état q' n'appartient pas à Q^1 (resp. Q^2), il appartient à Q^2 (resp. Q^1) ainsi que son prédécesseur. De plus, si un chemin sur $A_{1,2}$ part d'un état de Q^1 pour rejoindre un état de Q^2, alors il passe forcément par un état qui appartient à $Q^1 \cap Q^2$.
2. $\forall j \in \{1, 2\},\ \{q, q'\} \subseteq Q^j \Rightarrow (q, l, q') \in T^j$ (d'après la troisième propriété de la définition des chaînes d'automates).

Soit une trajectoire chem $= ((q_0, \ldots, q_m), (l_1, \ldots, l_m))$ de $\mathcal{E}_{A_{1,2}}$. Alors, chem est une trajectoire de la reconstruction non simplifiée $A^{ns}_{1,2}$ de $\mathcal{E}_{A_{1,2}}$ puisque :

- par définition toutes les transitions de la chaine sont des transitions de la reconstruction non simplifiée,
- l'état q_0 de la trajectoire appartient à $I^1 = I^{ns}_{1,2}$,
- l'état q_m de la trajectoire appartient à $F^2 = F^{ns}_{1,2}$.

Or, puisque chem est une trajectoire de $A^{ns}_{1,2}$, les états et transitions de chem sont également présents dans $A_{1,2}$ et chem est une trajectoire de $A_{1,2}$.

À présent, soit une trajectoire chem $= ((q_0, \ldots, q_m), (l_1, \ldots, l_m))$ de $A_{1,2}$. Soit k la plus petite valeur de $\{0, \ldots, m\}$ telle que $q_k \in Q^1 \cap Q^2$ (k existe d'après la remarque 1 ci-avant). $\forall i \leq k,\ q_i \in Q^1$ et $\forall i \leq k,\ (q_{i-1}, l_i, q_i) \in T^1$ d'après la remarque 2. chem^1 est donc un chemin sur A^1. De plus, $q_0 \in I^1$ et $q_k \in F^1$ (d'après la deuxième condition de la définition des chaînes d'automates[1]). Donc, chem$^1 = ((q_0, \ldots, q_k), (l_1, \ldots, l_k))$ est une trajectoire de A^1.

Nous démontrons maintenant par l'absurde que $\forall i > k,\ q_i \in Q^2$. Supposons qu'il existe j, la plus petite valeur telle que $j > k$ et $q_j \notin Q^2$. Alors $q_j \in Q^1$ et, d'après la remarque 1, $q_{j-1} \in Q^1 \cap Q^2$. On définit l la plus petite valeur de $\{j+1, \ldots, m\}$ telle que $q_l \in Q^1 \cap Q^2$ (l existe pour la même raison que k). Le chemin chem$' = ((q_{j-1}, \ldots, q_l), (l_j, \ldots, l_l))$ est donc un chemin A^1. Or, q_{j-1} et q_l appartiennant à $Q^1 \cap Q^2$, et d'après la troisième propriété de la définition de chaîne d'automates, on conclue que chem$'$ est un chemin de A^2 et que q_j est un état de A^2. Ceci est en contradiction avec l'existence de q_j. On voit donc que $\forall i > k,\ q_i \in Q^2$. Et de manière parallèle à chem^1, on voit que chem$^2 = ((q_k, \ldots, q_m), (l_{k+1}, \ldots, l_m))$ est une trajectoire de A^2.

chem est obtenu par reconstruction de chem^1 et chem^2. C'est donc une trajectoire de $\mathcal{E}_{A_{1,2}}$.

Nous avons prouvé que (A^1, A^2) est un découpage correct de $A_{1,2}$. Nous définissons $A_{1,i}$ récursivement par :

- $A_{1,1} = A_1$ et
- $\forall i \geq 2$, $A_{1,i}$ est la reconstruction de $\mathcal{E}_{A_{1,i}}$ où $\mathcal{E}_{A_{1,i}} = (A_{1,i-1}, A^i)$.

Nous démontrons maintenant par récurrence que les trajectoires de $A_{1,i}$ sont identiques aux trajectoires de $(A^1, \ldots, A^i)$.

- Prenons $i \geq 2$. Considérons que les trajectoires de $A_{1,i-1}$ sont identiques aux trajectoires de $(A^1, \ldots, A^{i-1})$. Alors, les trajectoires de $(A^1, \ldots, A^{i-1}, A^i)$ sont identiques aux trajectoires de $\mathcal{E}_{A_{1,i}} = (A_{1,i-1}, A^i)$. Or, $A_{1,i}$ est la reconstruction

[1] Ce point justifie la deuxième condition de la définition

de $\mathcal{E}_{A_{1,i}}$ et nous avons prouvé ci-dessus que leurs ensembles de trajectoires sont identiques. Ainsi, les trajectoires de $A_{1,i}$ et $(A^1, \ldots, A^i)$ sont identiques.

– Pour $i = 1$, le résultat est trivialement vrai.

Nous avons montré par récurrence que le résultat est vrai pour tout $i \geq 1$. D'autre part, par définition de reconstruction $A_R = A_{1,n}$. L'ensemble des trajectoires de A_R est donc identique à l'ensemble des trajectoires de $(A^1, \ldots, A^n)$.

Donc, puisque $\mathcal{E}_A$ est un découpage correct de A, $A = A_R$. □

Outre qu'elle prouve le théorème, cette preuve est intéressante puisqu'elle montre que la chaîne $(A_{1,i}, A^{i+1}, \ldots, A^n)$ est un découpage correct de A, $A_{1,i}$ étant un découpage correct de $(A^1, \ldots, A^i)$. Ce résultat peut être généralisé et il est possible de remplacer les $(j - i + 1)$ automates $A^i, \ldots, A^j$ d'une chaîne d'automates $(A^1, \ldots, A^n)$ par la reconstruction de la chaîne d'automates $(A^i, \ldots, A^j)$. On peut ainsi imaginer une reconstruction de $\mathcal{E}_A$ en plusieurs étapes. On peut également imaginer une généralisation des chaînes d'automates constituées d'automates et de chaînes d'automates. Ce point n'est cependant pas développé.

Puisque l'opération de reconstruction d'une chaîne d'automates peut se voir comme l'opération inverse du découpage, on décide de noter la reconstruction de la chaîne $\mathcal{E}_A$ de la manière suivante : $Sli^{-1}(\mathcal{E}_A)$. Remarquons que cette notation suggère que le découpage correct est une fonction bijective, ce qui n'est pas le cas. Il existe plusieurs découpages corrects d'un automate. En revanche, il n'existe qu'une seule reconstruction d'une chaîne d'automates. La reconstruction est donc effectivement une fonction.

Il est à présent possible de revenir sur la définition de chaîne d'automates. La deuxième condition indique que les états présents sur deux automates A^i et A^j de la chaîne doivent être des états *frontière* ; c'est-à-dire qu'ils doivent être présents sur tous les automates entre A^i et A^j et que ces états doivent être finaux quand ils sont présents dans un automate suivant et initiaux quand ils sont présents dans un automate précédent. Cette condition est nécessaire pour s'assurer qu'une trajectoire de A est la reconstruction de n trajectoires (voir la preuve précédente).

Par ailleurs, la troisième condition est également importante. Elle permet de s'assurer qu'une trajectoire sur l'automate reconstruit est présente sur la chaîne d'automates. Prenons l'exemple de la figure 3.3. Dans cet exemple, seul l'état q commun à deux morceaux est présenté (les transitions entre cet état et les autres sont en pointillés). On note $t_1 = (q, l_1, q)$ la transition bouclant sur q et étiquetée par l_1, et $t_2 = (q, l_2, q)$ avec $l_1 \neq l_2$. On a : $t_1 \in T^i \setminus T^{i+1}$ et $t_2 \in T^{i+1} \setminus T^i$. La troisième condition n'est donc ici pas respectée. Alors, les trajectoires sur la séquence d'automates indiquent que l_1 a eu lieu un nombre quelconque de fois puis l_2 a lieu un nombre quelconque de fois. En revanche, la reconstruction indique qu'il est possible que l_1 ait eu lieu après l_2. Ainsi, si la troisième condition n'est pas respectée, la reconstruction n'est pas une opération qui permette d'obtenir de manière sûre un automate comportant le même ensemble de trajectoires que la séquence d'automates.

Dans la définition de chaîne d'automates donnée dans [GCL05b], les conditions ne sont pas énoncées mais on considère qu'une chaîne d'automates n'est un découpage

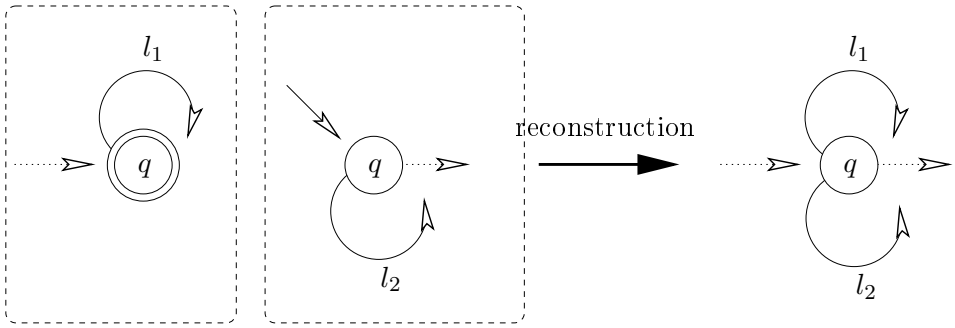

FIG. 3.3 – Démonstration de l'importance de la troisième condition de la définition de chaîne d'automates

correct que si la reconstruction donne un automate disposant des mêmes trajectoires que la chaîne. La définition qui nous avons donnée plus haut est plus restrictive mais permet d'être certain de l'existence des propriétés associées à la reconstruction.

3.2.2 Synchronisation de chaîne

Nous avons vu qu'il était possible de considérer un automate *par morceaux* grâce au formalisme des chaînes d'automates. Il est alors nécessaire de redéfinir les opérations utilisées par les automates dans le cadre du diagnostic pour les chaînes d'automates. La principale opération est la synchronisation d'automates. Nous présentons ici un premier résultat qui est *raffiné* dans 3.3.2. Pour cela, il est tout d'abord nécessaire de rappeler les fermetures préfixe et suffixe.

Définition 3.8 (Fermeture préfixe)**.**
Soit $A = (Q, E, T, I, F)$ un automate. La fermeture préfixe *de A est l'automate noté A^+ tel que tous les états sont finaux ($F^+ = Q^+ = Q$).*

Définition 3.9 (Fermeture suffixe)**.**
Soit $A = (Q, E, T, I, F)$ un automate. La fermeture suffixe *de A est l'automate noté A^- tel que tous les états sont initiaux ($I^- = Q^- = Q$).*

On note $A^\#$ l'automate calculé par fermeture suffixe et fermeture préfixe. $A^\# = A^{+^-} = A^{-^+}$.

Soit M un automate. Une trajectoire sur l'automate correspond à un comportement pendant la fenêtre globale. Considérons une fenêtre $\mathcal{W}$ qui ne commence pas forcément à t_0. Alors, l'état a pu évoluer entre t_0 et le début de la fenêtre $\mathcal{W}$. Ainsi, les trajectoires ne commencent pas nécessairement par un état initial de M, et il faut donc considérer l'automate M^-.

Lorsqu'on cherche à synchroniser le comportement de M et $\mathcal{E}_A$, on synchronise chaque morceau A^i avec M. Cependant, le morceau A^i (avec $i > 1$) correspond à une fenêtre qui ne commence pas nécessairement à t_0. Ainsi, la synchronisation doit s'effectuer entre A^i et M^-.

Pour une raison symétrique, il faut parfois considérer la fermeture préfixe de l'automate lors de la synchronisation de M et $\mathcal{E}_A$.

Définition 3.10 (Synchronisation d'un automate et d'une chaîne d'automates)**.**
Soit $\mathcal{E}_A = (A^1, \ldots, A^n)$ une chaîne d'automates et M un automate. La synchronisation *de M et $\mathcal{E}_A$ notée $M \otimes \mathcal{E}_A$ est une séquence d'automates $(A^1_\otimes, \ldots, A^n_\otimes)$ définie par :*

- $A^1_\otimes = M^+ \otimes A^1$,
- $\forall i \in \{2, \ldots, n-1\},\ A^i_\otimes = M^\# \otimes A^i$ *et*
- $A^n_\otimes = M^- \otimes A^n$.

Remarquons que si $n = 1$, alors $M \otimes \mathcal{E}_A = M \otimes (A^1) = M \otimes (A) = (M \otimes A)$.

La synchronisation de A et M se fait en utilisant l'ensemble des événements de synchronisation $E = E_A \cap E_M$. Aussi, il est nécessaire que le même ensemble de synchronisation soit utilisé pour tous les morceaux de la chaîne d'automates lors de la synchronisation de $\mathcal{E}_A$ et M : $\forall i, \forall M,\ E_{A^i} \cap E_M = E$. Ceci est la raison de la première condition de la définition de chaîne d'automates.

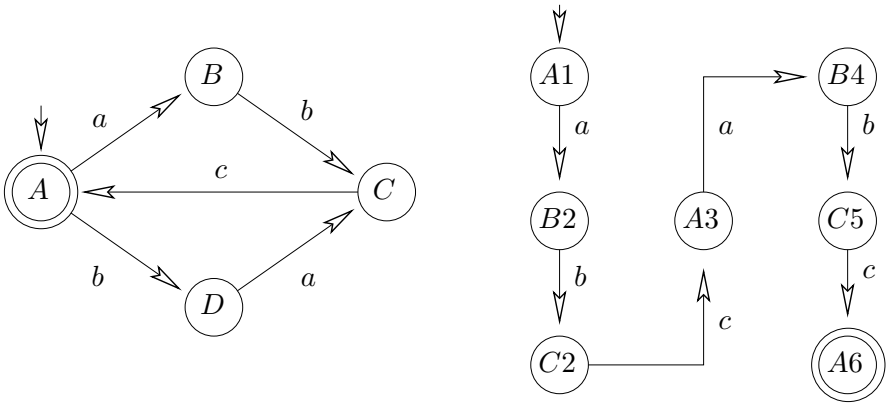

FIG. 3.4 – Automate M et la synchronisation de M avec l'automate de la figure 3.2

Considérons à présent l'automate M présenté sur la gauche de la figure 3.4. La partie droite de la figure présente par ailleurs la synchronisation $M \otimes A$ de M avec la reconstruction A (figure 3.2) de la chaîne d'automates $\mathcal{E}_A$ (figure 3.1). La synchronisation $M \otimes \mathcal{E}_A$ de M et $\mathcal{E}_A$ est présentée figure 3.5. On voit que la chaîne produite comporte de nombreux états inutiles ($D7$, $A8$, etc.) qui sont représentés sur la figure de manière grisée. Ces états sont en partie dûs aux états inutiles de la chaîne d'automates $\mathcal{E}_A$ (voir par exemple l'état $D7$), mais pas seulement (voir par exemple l'état $D3$). Par la suite (en partie 3.3), nous voyons comment réduire ou supprimer complètement ces états inutiles. Nous pouvons remarquer que la chaîne construite $M \otimes \mathcal{E}_A$ est un découpage correct de $M \otimes A$. Par la suite, nous démontrons que ce résultat est toujours vrai.

Théorème 3.3.
Soit $\mathcal{E}_A$ une chaîne d'automates et M un automate. Alors, $M \otimes \mathcal{E}_A$ est une chaîne d'automates.

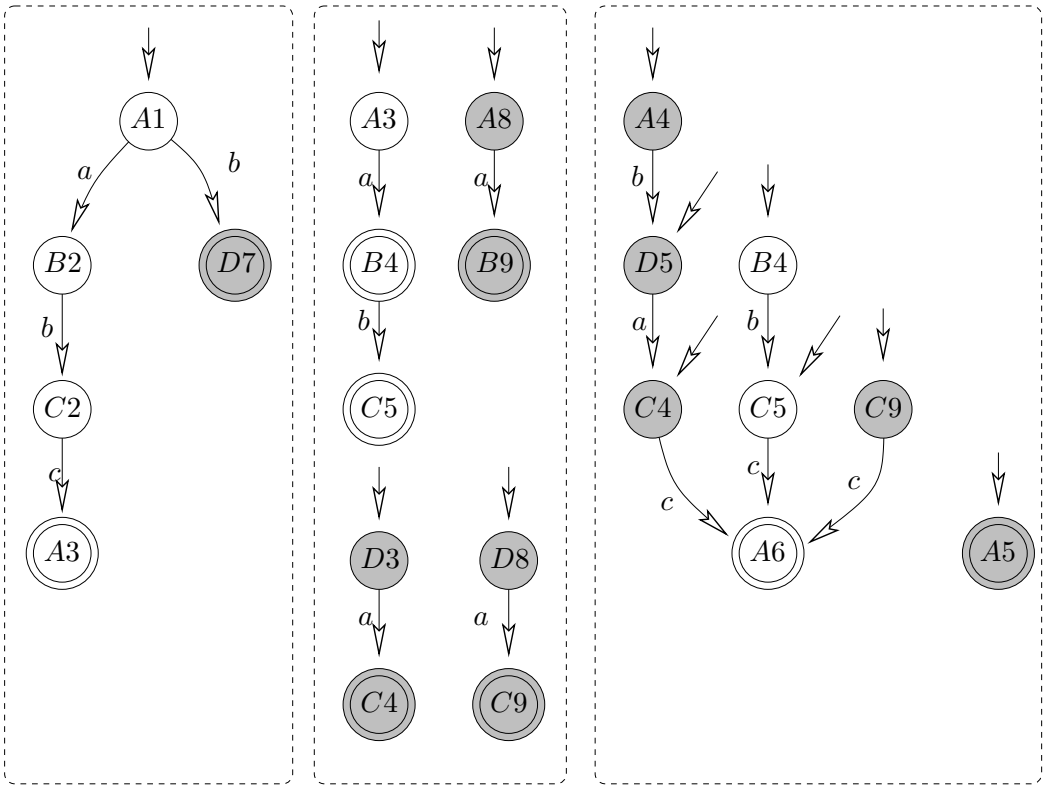

FIG. 3.5 – Synchronisation de l'automate de la figure 3.4 et de la chaîne d'automates de la figure 3.1

Démonstration. Pour cette preuve, nous prenons la notation suivante :

- $M = (Q_M, E_M, T_M, I_M, F_M)$,
- $\mathcal{E}_A = (A^1, \ldots, A^n)$ avec $A^i = (Q^i, E^i, T^i, I^i, F^i)$ et $E^i = E$,
- $\mathcal{E}_A \otimes M = (A^1_\otimes, \ldots, A^n_\otimes)$ avec $A^i_\otimes = (Q^i_\otimes, E^i_\otimes, T^i_\otimes, I^i_\otimes, F^i_\otimes)$, $A^i_\otimes$ non simplifié.

Considérons les trois conditions de la définition de chaîne d'automates :

- $\forall i, j,\ E^i_\otimes = E^j_\otimes = E_M \cup E$. Cette condition est vérifiée.
- $\forall i, j,\ j > i, \forall q_\otimes,\ q_\otimes \in Q^i_\otimes \cap Q^j_\otimes \Rightarrow q_\otimes \in F^i_\otimes \wedge q_\otimes \in I^{i+1}_\otimes$. Si $q_\otimes = (q_M, q) \in Q^i_\otimes$, alors $q_M \in Q_M$ et $q \in Q^i$. De même, $q \in Q^j$. Ainsi, $q \in F^i$ et $q \in I^{i+1}$ (puisque $\mathcal{E}_A$ est une chaîne d'automates). De plus, $q_M \in F^+$ et $q_M \in F^\#$. Or $i \neq 1$ implique $A^i_\otimes = M^+ \otimes A^i$ ou $A^i_\otimes = M^\# \otimes A^i$. Donc, $q^i \in F^i$. Parallèlement, $q^i \in I^{i+1}$. Cette condition est donc vérifiée.
- $\forall i, j, \forall q_\otimes, q'_\otimes$, si $\{q_\otimes, q'_\otimes\} \subseteq Q^i_\otimes \cap Q^j_\otimes$ alors $\forall \text{chem}_\otimes$, chemin de $A^i_\otimes$ entre $q_\otimes$ et $q'_\otimes$, $\text{chem}_\otimes$ est aussi un chemin sur $A^j_\otimes$. Soit un chemin $\text{chem}_\otimes = (((q_0, q_{M0}), \ldots, (q_m, q_{Mm}))(l_1, \ldots, l_m))$ sur $A^i\otimes$. Alors, $\text{chem} = ((q_0, \ldots, q_m), (l_1, \ldots, l_m))$ est un chemin sur A^i (par définition de la synchronisation), $q_0 \in Q^j$ et $q_m \in Q^j$. Donc, chem est un chemin de A^j. De plus, $\text{chem}_M = ((q_{M0}, \ldots, q_{Mm}), (l_{M1}, \ldots, l_{Mm}))$ est un chemin sur M. Donc, $\text{chem}_\otimes$

est un chemin sur $A^j_{\otimes}$. Cette condition est donc vérifiée.

□

Théorème 3.4.
Soit $\mathcal{E}_A$ une chaîne d'automates. Soit M un automate. Alors, $M \otimes \mathcal{E}_A$ est un découpage correct de $M \otimes Sli^{-1}(\mathcal{E}_A)$.

Démonstration. Nous reprenons la notation précédente. Soit $\text{chem}' = ((q'_0, \ldots, q'_m), (l'_1, \ldots, l'_m))$ une trajectoire de $M \otimes Sli^{-1}(\mathcal{E}_A)$ telle que $\forall i,\ q'_i = (q_i, q_i^M)$ et $l'_i = \Theta_.(l_i, l_i^M,)$ Alors, $\text{chem} = ((q_0, \ldots, q_m), (l_1, \ldots, l_m))$ est une trajectoire de $Sli^{-1}(\mathcal{E}_A)$. Donc, chem est la reconstruction de n trajectoires $\text{chem}^i = ((q_{f(i)}, \ldots, q_{f(i+1)}), (l_{f(i)+1}, \ldots, l_{f(i+1)}))$ trajectoire de A^i. De même $\text{chem}^M = ((q_0^M, \ldots, q_m^M), (l_1^M, \ldots, l_m^M))$ est une trajectoire de M et $\text{chem}_i^M = ((q_{f(i)}^M, \ldots, q_{f(i+1)}^M), (l_{f(i)+1}^M, \ldots, l_{f(i+1)}^M))$ est une trajectoire de $M^{\#}$ (de M^+ pour $i = 1$, M^- pour $i = n$). Ainsi, $\text{chem}'^i = ((q'_{f(i)}, \ldots, q'_{f(i+1)}), (l'_{f(i)+1,\ldots,l'_{f(i+1)}}))$ est une trajectoire de $A^i_{\otimes}$. Donc, chem′, la reconstruction des n trajectoires chem'^i est une trajectoire de $M \otimes \mathcal{E}_A$.

De la même manière, toute trajectoire de $M \otimes \mathcal{E}_A$ est une trajectoire de $M \otimes Sli^{-1}(\mathcal{E}_A)$. □

Ce résultat est très important. En effet, considérons un automate A que, pour des raisons de commodité (dont certaines sont présentées par la suite), on désire représenter sous la forme d'une chaîne d'automates $\mathcal{E}_A$. Si $\mathcal{E}_A$ est un découpage correct de A, alors il est possible de calculer une chaîne d'automates qui est le découpage correct de $M \otimes A$ sans reconstruire l'automate A.

L'application de ce résultat au diagnostic est maintenant présentée.

3.2.3 Application au diagnostic

Les résultats présentés ci-dessus peuvent être appliqués au diagnostic. En particulier, on peut considérer non pas un automate des observations mais une chaîne d'automates des observations et effectuer un calcul à l'aide de cette chaîne.

Nous utilisons tout d'abord les chaînes d'automates pour le diagnostic dans un contexte non incrémental, puis dans un contexte incrémental.

3.2.3.1 Calcul non incrémental du diagnostic

Les résultats présentés ci-dessus permettent une définition de diagnostic par morceaux.

Définition 3.11 (Diagnostic par morceaux)**.**
Soit Mod l'automate représentant le modèle du système. Soit Obs l'automate des observations et $\mathcal{E}_{Obs}$ un découpage correct de Obs. Le diagnostic par morceaux *de Obs est défini comme suit : $\mathcal{E}_\Delta = Mod \otimes \mathcal{E}_{Obs}$.*

Le diagnostic par morceaux est donc une chaîne d'automates $\Delta = (\Delta^1, \ldots, \Delta^n)$ où Δ^i est appelé le morceau numéro i de diagnostic.

Résultat 3.1.
Le diagnostic par morceaux $\mathcal{E}_\Delta$ est un découpage correct du diagnostic Δ.

Ce résultat est intéressant pour deux raisons.

1. Tout d'abord, dans certains cas, il est plus facile d'exprimer l'automate des observations sous forme de chaînes d'automates notamment pour le diagnostic incrémental, et encore plus dans le cas du diagnostic en-ligne comme nous le voyons par la suite. Remarquons de plus que les morceaux de l'automate étant petits, ils sont plus faciles à manipuler que l'automate des observations. Aussi, cette méthode permet de calculer le diagnostic sans passer par la construction de l'automate global des observations du système. De plus, si on considère que, pour les mêmes raisons, il est plus pratique de représenter le diagnostic sous forme de chaîne, alors cette méthode le permet.
2. Ensuite, grâce à ce résultat, il est possible de calculer les morceaux de diagnostic chacun séparément et en parallèle.

Il est possible d'appliquer ce résultat pour calculer le diagnostic de manière incrémentale.

3.2.3.2 Calcul incrémental du diagnostic

Soit $i+1$ automates $Obs_1, \ldots, Obs_{i+1}$ tels que $\forall k \in \{1, \ldots, i\}$, Obs_k est un préfixe de Obs_{k+1}. Considérons qu'il existe $Obs^1, \ldots, Obs^{i+1}$ tel que $\forall k \in \{1, \ldots, i+1\}$, $(Obs^1, \ldots, Obs^k)$ est un découpage correct de Obs_k.

On note $\mathcal{E}_{\Delta_i} = (\Delta^1, \ldots, \Delta^i)$ le diagnostic par morceaux de Obs_i. $\mathcal{E}_{\Delta_i}$ est un découpage correct de $\Delta_i = Mod \otimes Obs_i$. On note $\mathcal{E}_{\Delta_{i+1}}$ le diagnostic par morceaux de Obs_{i+1}. Alors, on a le résultat suivant :

Résultat 3.2.
$\mathcal{E}_{\Delta_{i+1}} = (\Delta^1, \ldots, \Delta^i, \Delta^{i+1})$ *où* $\Delta^{i+1} = Mod^\# \otimes Obs^{i+1}$.

Démonstration. On a : $\mathcal{E}_{\Delta_{i+1}} = Mod \otimes (Obs^1, \ldots, Obs^{i+1}) = (Mod^+ \otimes Obs^1, Mod^\# \otimes Obs^2, \ldots, Mod^\# \otimes Obs^i, Mod^- \otimes Obs^{i+1})$.

Or $\mathcal{E}_{\Delta_i} = Mod \otimes (Obs^1, \ldots, Obs^i) = (Mod^+ \otimes Obs^1, Mod^\# \otimes Obs^2, \ldots, Mod^- \otimes Obs^i)$. De plus, $Mod^- = Mod^\#$ puisque $F^{Mod} = Q^{Mod}$.

Donc, $\mathcal{E}_{\Delta_{i+1}} = (\Delta^1, \ldots, \Delta^i, Mod^\# \otimes Obs^{i+1})$. □

Ce résultat indique qu'il suffit uniquement de calculer le dernier morceau de diagnostic pour obtenir le diagnostic de Δ_{i+1} à partir de Δ_i.

Cependant, comme présenté par la figure 3.5, les morceaux issus d'une synchronisation comportent de nombreux états inutiles. Ainsi, dans la figure 3.5, il n'est pas évident de déterminer quels événements ont effectivement eu lieu. Par exemple, il n'est pas facile de déterminer si le système est passé ou non par l'état D. Aussi, l'opération de reconstruction est nécessaire pour supprimer ces états inutiles. La section suivante propose une extension permettant de supprimer ces états inutiles.

3.3 Extension : le diagnostic incrémental

Nous avons vu qu'une chaîne d'automates peut disposer d'un certain nombre d'états qui pourraient être ôtés sans dommage. Ces états sont qualifiés d'*inutiles*

Définition 3.12 (État inutile).
Un état q d'une chaîne d'automates $\mathcal{E}_A$ est dit inutile *si aucune trajectoire de $\mathcal{E}_A$ ne passe par cet état.*

La suppression des états inutiles des chaînes d'automates est un point indispensable pour l'utilisation des chaînes d'automates dans le cadre du diagnostic. Cette opération est appelée *raffinement*. L'opération de raffinement a été introduite dans [LC00]. Raffiner consiste à supprimer des états d'un automate grâce à une information obtenue dans la fenêtre précédente ou dans la fenêtre suivante. Cette notion ne doit pas être confondue avec la notion de raffinement définie comme l'opération inverse de l'abstraction. Grâce au raffinement, il est possible de modifier la définition de synchronisation d'une chaîne et d'un automate et réduire le nombre d'états produits. Nous présentons alors une nouvelle opération de synchronisation, la *synchronisation incrémentale*. Cette synchronisation est ensuite appliquée au diagnostic. Ensuite, nous discutons de l'intérêt d'une synchronisation encore plus restrictive appelée synchronisation raffinée.

3.3.1 Raffinement de chaîne

Nous présentons à présent une technique permettant de mettre en évidence les états inutiles d'une chaîne d'automates et permettant de s'en débarrasser. Ces propriétés sont utilisées par la suite notamment dans le cadre de la synchronisation incrémentale.

On voit sur l'exemple de la chaîne d'automates de la figure 3.1 que l'état 7 ne peut être sur aucune trajectoire de la chaîne d'automates parce qu'il s'agit d'un état final du premier automate de la chaîne et que cet état n'apparaît pas dans le deuxième automate. L'état 8 est inutile pour une raison symétrique. L'état 9 est justifié par l'existence de l'état 8.

Ces constats permettent de donner les définitions suivantes :

Définition 3.13 (Chaîne I-raffinée).
Une chaîne d'automates $\mathcal{E}_A = (A^1, \ldots, A^n)$ est dite I-raffinée *si elle vérifie la propriété suivante : $\forall i \in \{1, \ldots, n-1\},\ I^{i+1} \subseteq F^i$.*

On voit que la chaîne d'automates de la figure 3.1 n'est pas I-raffinée puisque l'état $8 \in I^2$ et $8 \notin F^1$.

Nous prenons l'hypothèse que tous les automates de la chaîne $\mathcal{E}_A = (A^1, \ldots, A^n)$ I-raffinée sont simplifiés. Alors, pour tout état q de l'automate A^i, il existe un chemin menant d'un état $q_0 \in I^i$ à l'état q. Grâce à la propriété de I-raffinement, on peut facilement démontrer récursivement qu'il existe un chemin entre $q_0 \in I^1$ et q sur la chaîne d'automate. Soit q un état de A^n. Alors, il existe un chemin menant de $q_0 \in I^1$ à q grâce à l'hypothèse de I-raffinement et un chemin menant de q à $q_m \in F^n$ grâce à

l'hypothèse d'automates simplifiés. Ainsi, pour tout état q de A^n, il existe une trajectoire passant par q. Donc, aucun état de A^n est inutile.

Dans certains contextes, seul le dernier automate de la chaîne est intéressant. En effet, lorsqu'on veut connaître l'état final du système à l'aide d'un diagnostic par morceaux, ou si l'on cherche à connaître les derniers comportements qui ont eu lieu sur le système, seul le dernier automate nous intéresse. Les autres automates permettent d'indiquer quels sont les états inutiles du dernier automate. Dans le cas d'une chaîne d'automates I-raffinée, on voit que les précédents automates ne sont pas utiles puisque le dernier automate ne comporte aucun état inutile.

Rappelons ainsi que la définition de reconstruction indique que l'automate reconstruit A est tel que l'ensemble des états finaux F est l'ensemble des états finaux F^n du dernier automate de la chaîne. Cependant, dans le cas général, la simplification de la chaîne reconstruite supprime certains états finaux. Ainsi, on a $F \subseteq F^n$. Dans le cas d'une chaîne I-raffinée, l'ensemble des états finaux de la reconstruction de la chaîne est exactement l'ensemble des états finaux du dernier automate : $F = F^n$.

Définition 3.14 (Chaîne F-raffinée)**.**
Une chaîne d'automates $\mathcal{E}_A = (A^1, \ldots, A^n)$ *est dite* F-raffinée *si elle vérifie la propriété suivante :* $\forall i \in \{1, \ldots, n-1\},\ F^i \subseteq I^{i+1}$.

On voit que la chaîne d'automates de la figure 3.1 n'est pas F-raffinée puisque l'état $7 \in F^1$ et $7 \notin I^2$.

Une chaîne F-raffinée est intéressante puisque si on se déplace sur la chaîne en partant de n'importe quel état, on est assuré de trouver un chemin qui mène à un état final du dernier automate. Par exemple, dans la chaîne d'automates *non* F-raffinée de la figure 3.1, si on atteint l'état 7 du premier automate, il ne nous est plus possible d'atteindre un état final. Ce cas de figure ne pourrait pas arriver dans une chaîne F-raffinée.

Définition 3.15 (Chaîne raffinée)**.**
Une chaîne d'automates est raffinée si elle est à la fois I-raffinée et F-raffinée.

L'intérêt d'une chaîne raffinée est qu'aucun état de la chaîne n'est inutile.

Nous présentons à présent les opérations permettant de supprimer les états inutiles.

Définition 3.16 (I-raffinement)**.**
Le I-raffinement *d'une chaîne* $\mathcal{E}_A = (A^1, \ldots, A^n)$*, défini si* $\mathcal{E}_A$ *n'est pas I-raffinée, est une séquence d'automates* $\mathcal{E}'_A = (A'^1, \ldots, A'^n)$ *telle que* $\exists i \in \{2, \ldots, n\}$ *et* $\exists q \in I^i \setminus F^{i-1}$ *et :*

- $\forall j \neq i,\ A'^j = A^j$ *et*
- A'^i *est l'automate simplifié de* $(Q^i, E^i, T^i, I^i \setminus \{q\}, F^i)$.

Le I-raffinement est une opération consistant à supprimer de l'ensemble des états initiaux d'un automate A^i l'état q qui n'est pas final dans le précédent automate A^{i-1}. Par exemple, un I-raffinement de la chaîne d'automates de la figure 3.1 permet de

retirer l'état 8 de l'ensemble des états initiaux du deuxième automate de la chaîne. Par simplification de l'automate A'^2, les états 8 et 9 sont alors retirés du deuxième automate. Un second I-raffinement permet d'ôter l'état 9 du troisième automate de la chaîne. La chaîne d'automates obtenue est présentée sur la figure 3.6.

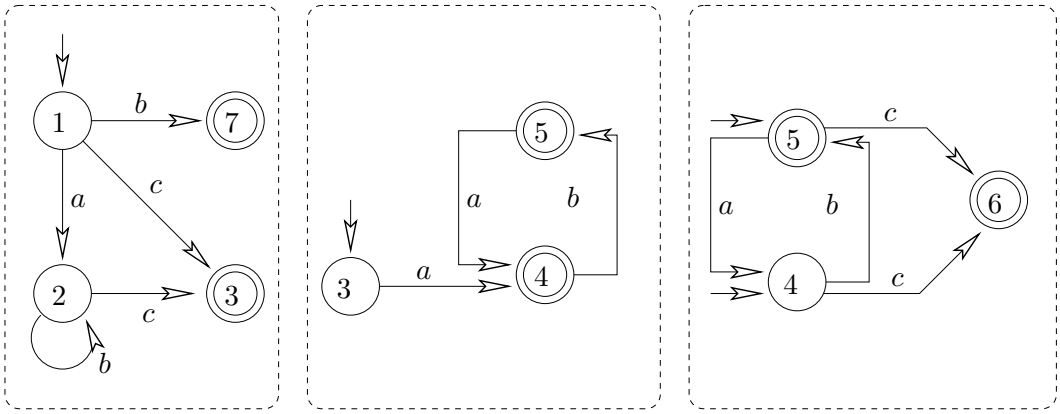

FIG. 3.6 – Chaîne d'automates I-raffinée obtenue par I-raffinements successifs de la chaîne de la figure 3.1

Définition 3.17 (F-raffinement)**.**
Le F-raffinement *d'une chaîne* $\mathcal{E}_A = (A^1, \ldots, A^n)$*, défini si* $\mathcal{E}_A$ *n'est pas F-raffinée, est une séquence d'automates* $\mathcal{E}'_A = (A'^1, \ldots, A'^n)$ *telle que* $\exists i \in \{1, \ldots, n-1\}$ *et* $\exists q \in F^i \setminus I^{i+1}$ *et :*

- $\forall j \neq i,\ A'^j = A^j$ *et*
- A'^i *est l'automate simplifié de* $(Q^i, E^i, T^i, I^i, F^i \setminus \{q\})$.

Le F-raffinement est l'opération symétrique au I-raffinement. L'opération de F-raffinement de la chaîne d'automates de la figure 3.1 permet ainsi de supprimer l'état inutile 7 de la chaîne d'automates.

On utilise le terme générique de raffinement pour désigner un I-raffinement ou un F-raffinement. Le raffinement est une opération qui ne supprime pas de trajectoires.

Théorème 3.5.
Soit $\mathcal{E}_A$ *une chaîne d'automates. La séquence d'automates* $\mathcal{E}'_A$ *obtenue par raffinement de* $\mathcal{E}_A$ *est une chaîne d'automates. De plus, l'ensemble des trajectoires de* $\mathcal{E}'_A$ *est égal à l'ensemble des trajectoires de* $\mathcal{E}_A$*.*

Démonstration. Considérons un I-raffinement de $\mathcal{E}_A$ en $\mathcal{E}'_A$ ($\mathcal{E}'_A$ étant la chaîne d'automates *non simplifiés*), la démonstration pour le F-raffinement est symétrique. Soit q l'état ôté de I^i. Considérons les trois conditions de la définition de chaîne d'automates :

- $\forall i, j,\ E'^i = E'^j$. Cette condition est vérifiée.
- $\forall i, j,\ j > i, \forall q,\ q \in Q'^i \cap Q'^j \Rightarrow q \in F'^i \wedge q \in I'^{i+1}$. L'état q n'appartient pas à F^{i-1}. Puisque $q \in I^i$, l'état q n'appartient pas à Q^{i-1}. Donc, $i \notin Q'^{i-1}$. La

propriété est donc vérifiée pour q. Aucun autre état n'est retiré des ensemble I^j ou F^j ni ajouté à Q^j. La condition est donc vérifiée.
- $\forall i, j, \forall q, q'$, si $\{q, q'\} \subseteq Q'^i \cap Q'^j$ alors $\forall p$, chemin de A'^i entre q et q', p est aussi un chemin sur A'^j. $\forall j,\ Q^j = Q'^j$ et $T^j = T'^j$. Ainsi, si la condition est vérifiée pour $\mathcal{E}_A$, cette condition est également vérifiée pour $\mathcal{E}'_A$.

$\mathcal{E}'_A$ est donc une chaîne d'automates, ainsi que la chaîne d'automates où A'^i a été simplifié.

Considérons à présent l'ensemble de trajectoires $\mathcal{E}_A$ et $\mathcal{E}'_A$. Il est immédiat que toute trajectoire de $\mathcal{E}'_A$ est une trajectoire de $\mathcal{E}_A$.

Soit chem une trajectoire de $\mathcal{E}_A$. Alors, chem est la reconstruction des n trajectoires $(\text{chem}^1, \ldots, \text{chem}^n)$, avec $\forall j,\ \text{chem}^j = ((q_0^j, \ldots, q_{mj}^j), (l_1^j, \ldots, l_{mj}^j))$. Pour $j \neq i$, chem^j est une trajectoire de $A'^j = A^j$. Or $q \notin F^{i-1}$. Donc $q \neq q_0^i$ (puisque $q_0^i = q_m^{i-1}(i-1) \in F^{i-1}$). Ainsi chem^i est également une trajectoire de A'^i. Donc, chem est une trajectoire $\mathcal{E}'_A$. Les ensembles des trajectoire de $\mathcal{E}_A$ et $\mathcal{E}'_A$ sont donc identiques. □

3.3.2 Synchronisation incrémentale

La synchronisation incrémentale utilise les résultats précédents pour fournir une chaîne d'automates I-raffinée.

Définition 3.18 (I-restriction)**.**
Soit $A = (Q, E, T, I, F)$ un automate. La I-restriction *de A sur I' notée $A[I']$ est l'automate simplifié de $(Q, E, T, I \cap I', F)$.*

La I-restriction consiste à réduire le nombre d'états initiaux. Par la suite, par abus de langage, nous désignons la I-restriction sous le nom de restriction.

Nous donnons une définition de synchronisation incrémentale où nous restreignons l'ensemble des états initiaux de chaque morceau de la chaîne construite sur l'ensemble des états finaux du précédent morceau.

Définition 3.19 (Synchronisation incrémentale)**.**
La synchronisation incrémentale *de l'automate M et de la chaîne $\mathcal{E}_A$, notée $M \odot \mathcal{E}_A$ est la séquence d'automates $(A_\odot^1, \ldots, A_\odot^n)$ définie par :*
- *$A_\odot^1 = M^+ \otimes A^1$ si $n \neq 1$, $A_\odot^1 = M \otimes A^1$ sinon,*
- *$\forall i \in \{2, \ldots, n-1\}$, $A_\odot^i = (M^\# \otimes A^i)[F_\odot^{i-1}]$ où $F_\odot^{i-1}$ est l'ensemble des états finaux de $A_\odot^{i-1}$,*
- *si $n \neq 1$, $A_\odot^n = (M^- \otimes A^n)[F_\odot^{n-1}]$.*

Remarquons que l'automate $(M^\# \otimes A^i)[F_\odot^{i-1}]$ n'est pas nécessairement calculé par la synchronisation de $M^\#$ et A^i suivie d'une I-restriction, mais que les deux opérations peuvent être effectuées en même temps de manière efficace (voir section 7.5).

Cette opération est appelée *incrémentale* puisque le calcul est effectué de manière incrémentale, c'est-à-dire que l'automate $A_\odot^i$ est calculé en utilisant $A_\odot^{i-1}$. La figure 3.7 présente la synchronisation incrémentale de l'automate M de la figure 3.4 et de la chaîne d'automates $\mathcal{E}_A$ de la figure 3.1. On voit qu'on obtient une chaîne d'automates qui est un découpage de la synchronisation de M et de la reconstruction de $\mathcal{E}_A$. C'est ce que

nous prouvons par la suite. La chaîne d'automates est par ailleurs beaucoup plus petite que celle de la figure 3.5, et comporte beaucoup moins d'états inutiles. Il est difficile de quantifier ce gain qui dépend de la forme des automates du modèle et des observations. On peut cependant remarquer que si le nombre d'état du modèle est important et si la longueur des trajectoires les plus petites sur la chaîne d'automates est faible, alors ce gain devrait être important.

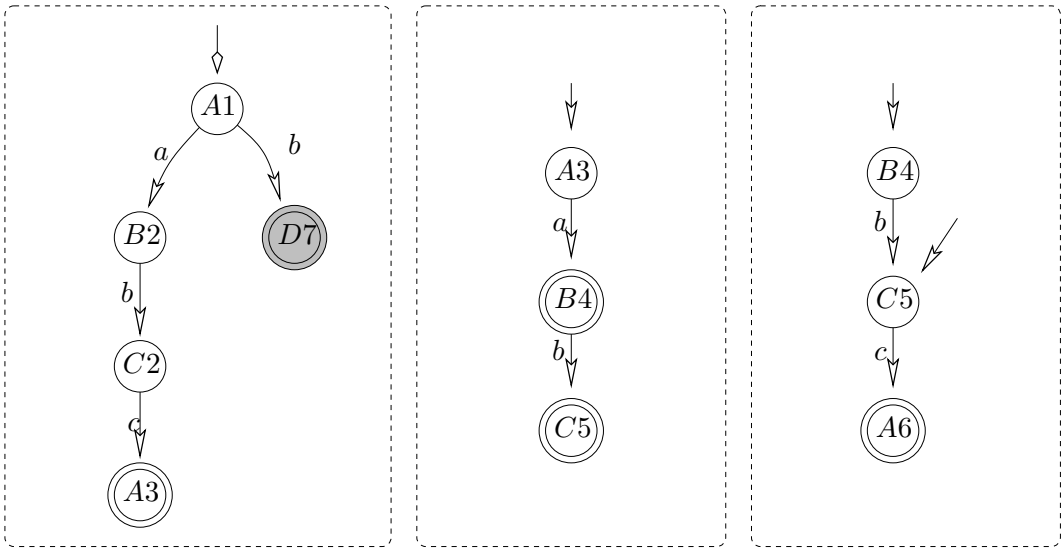

FIG. 3.7 – Synchronisation incrémentale de l'automate de la figure 3.4 et de la chaîne d'automates de la figure 3.1

Théorème 3.6.
Soit M un automate et $\mathcal{E}_A$ une chaîne d'automates. Alors, $M \odot \mathcal{E}_A$ est la chaîne d'automates I-raffinée obtenue par I-raffinements successifs de $M \otimes \mathcal{E}_A$.

Démonstration. Un I-raffinement supprime un état de la liste des états initiaux d'un morceau d'automate lorsque cet état n'appartient pas à l'ensemble des états finaux du morceau d'automate précédent. Ainsi, restreindre un morceau d'automate sur l'ensemble des états finaux du précédent morceau est équivalent à l'application successive de I-raffinements. Ainsi, $M \odot \mathcal{E}_A$ peut être obtenu par I-raffinements successifs de $M \otimes \mathcal{E}_A$. Par ailleurs, d'après le théorème 3.5, $M \odot \mathcal{E}_A$ est une chaîne d'automates.

Puisque le morceau numéro i de la chaîne $M \odot \mathcal{E}_A$ est restreint par l'ensemble des états finaux du $(i-1)$ème morceau, la chaîne $M \odot \mathcal{E}_A$ est I-raffinée. □

Corollaire 3.1.
Soit M un automate et $\mathcal{E}_A$ une chaîne d'automates. Alors $M \odot \mathcal{E}_A$ est un découpage correct de $M \otimes Sli^{-1}(\mathcal{E}_A)$.

Démonstration. Ce corollaire est le résultat du théorème 3.5 et du théorème 3.6. □

Ce résultat indique que la synchronisation incrémentale peut être utilisée pour calculer la synchronisation d'un automate et d'une chaîne d'automates à la place de la synchronisation présentée dans la section 3.2.

3.3.3 Application au diagnostic

Il est possible d'appliquer ce résultat directement au diagnostic.

3.3.3.1 Définition du diagnostic incrémental

Le diagnostic incrémental utilise la synchronisation incrémentale.

Définition 3.20 (Diagnostic incrémental)**.**
Soit Mod l'automate représentant le modèle du système. Soit Obs l'automate des observations et $\mathcal{E}_{Obs}$ un découpage correct de Obs. Le diagnostic incrémental *du système est défini comme suit : $\mathcal{E}_\Delta = Mod \odot \mathcal{E}_{Obs}$.*

Résultat 3.3.
Le diagnostic incrémental $\mathcal{E}_\Delta$ est un découpage correct du diagnostic Δ.

La chaîne de diagnostics obtenue par cette méthode est I-raffinée. L'avantage par rapport au résultat 3.1 est que l'ensemble des états finaux du dernier automate de la chaîne est l'ensemble des états finaux de la reconstruction de la chaîne. Ainsi, si le but du diagnostic est de connaître précisément l'ensemble des états du système à l'issue du diagnostic, ce calcul est suffisant. En revanche, l'utilisation de la synchronisation incrémentale ne permet plus un calcul en parallèle puisque l'ensemble des états finaux d'un morceau de diagnostic est nécessaire au calcul du morceau suivant.

Nous présentons à présent un calcul de manière incrémentale du diagnostic.

3.3.3.2 Calcul incrémental du diagnostic

Nous faisons une distinction entre le calcul incrémental du diagnostic et le diagnostic incrémental. Le diagnostic incrémental est une représentation du diagnostic sous forme de chaîne d'automates, tandis que le calcul incrémental du diagnostic consiste à utiliser un calcul précédemment effectué pour accélérer le calcul du diagnostic. Remarquons que le diagnostic incrémental est particulièrement adapté au calcul incrémental du diagnostic.

Soit $Obs_1, \ldots, Obs_{i+1}$, $i+1$ automates tels que $\forall k \in \{1, \ldots, i\}$, Obs_k est un préfixe de Obs_{k+1}. Considérons qu'il existe $Obs^1, \ldots, Obs^{i+1}$ tel que $\forall k \in \{1, \ldots, i+1\}$, $(Obs^1, \ldots, Obs^k)$ est un découpage correct de Obs_k.

On note $\mathcal{E}_{\Delta_i} = (\Delta^1, \ldots, \Delta^i)$ le diagnostic incrémental de Obs_i. $\mathcal{E}_{\Delta_i}$ est un découpage correct de $\Delta_i = Mod \otimes Obs_i$. On note $\mathcal{E}_{\Delta_{i+1}}$ le diagnostic incrémental de Obs_{i+1}. Alors, on a le résultat suivant :

Résultat 3.4.
$\mathcal{E}_{\Delta_{i+1}} = (\Delta^1, \ldots, \Delta^i, \Delta^{i+1})$ *où* $\Delta^{i+1} = (Mod^\# \otimes Obs^{i+1})[F^i_\Delta]$ *avec* F^i_Δ *l'ensemble des états finaux de* Δ^i.

Démonstration.
On a par définition :

$\mathcal{E}_{\Delta_i} = Mod \odot \mathcal{E}_{Obs_i} = ((Mod^+ \otimes Obs^1), (Mod^\# \otimes Obs^2)[F_\Delta^1], \ldots, (Mod^\# \otimes Obs^{i-1})[F_\Delta^{i-2}], (Mod^- \otimes Obs^i)[F_\Delta^{i-1}])$.

Par ailleurs, on a :

$\mathcal{E}_{\Delta_{i+1}} = Mod \odot \mathcal{E}_{Obs_{i+1}} = ((Mod^+ \otimes Obs^1), (Mod^\# \otimes Obs^2)[F_\Delta^1], \ldots, (Mod^\# \otimes Obs^{i-1})[F_\Delta^{i-2}], (Mod^\# \otimes Obs^i)[F_\Delta^{i-1}], (Mod^- \otimes Obs^{i+1})[F_\Delta'^i])$ où $F_\Delta'^i$ est l'ensemble des états finaux de $(Mod^\# \otimes Obs^i)[F_\Delta^{i-1}]$.

On a donc $\mathcal{E}_{\Delta_{i+1}} = (\Delta^1, \ldots, \Delta^{i-1}, (Mod^\# \otimes Obs^i)[F_\Delta^{i-1}], (Mod^- \otimes Obs^{i+1})[F_\Delta'^i])$. Or, $Mod^- = Mod^\#$ puisque tous les états de Mod sont finaux, comme nous l'avons indiqué en section 2.1, page 48. Ainsi, $(Mod^\# \otimes Obs^i)[F_\Delta^{i-1}] = (Mod^- \otimes Obs^i)[F_\Delta^{i-1}] = \Delta_i$ et $F_\Delta'^i = F_\Delta^i$. Donc, $\mathcal{E}_{\Delta_{i+1}} = (\Delta^1, \ldots, \Delta^i, \Delta^{i+1})$. □

Ce résultat permet de calculer le diagnostic de manière incrémentale comme présenté dans la définition 3.2. Étant donné $\mathcal{E}_{\Delta_i}$, le calcul de $\mathcal{E}_{\Delta_{i+1}}$ ne nécessite que le calcul de Δ^{i+1}. Ainsi, si on considère que chaque automate Obs^k a une taille comparable, alors le calcul de chaque $\mathcal{E}_{\Delta_{i+1}}$ à partir de $\mathcal{E}_{\Delta_i}$ nécessite une puissance de calcul identique pour tout i, et peut donc être envisagé en-ligne. Remarquons que l'hypothèse que chaque Obs^i ait une taille équivalente est une hypothèse importante.

Le diagnostic du système a deux buts : le premier est de trouver le comportement passé du système, le second est de trouver l'état actuel du système. Le résultat suivant est intéressant concernant le second point.

Résultat 3.5.
Dans le diagnostic incrémental $\mathcal{E}_{\Delta_i} = (\Delta^1, \ldots, \Delta^i)$, l'ensemble des états finaux de Δ_i est l'ensemble des états finaux de $\Delta^i = Sli^{-1}(\mathcal{E}_{\Delta_i})$.

Démonstration. Le diagnostic incrémental $\mathcal{E}_{\Delta_i} = (\Delta^1, \ldots, \Delta^i)$ est obtenu par synchronisation incrémentale. Or, la synchronisation incrémentale produit une chaîne d'automates I-raffinée (théorème 3.6). Ainsi, pour tout état q de la chaîne d'automate, il existe un chemin partant d'un état initial de Δ^1 et menant à cet état q. Par ailleurs, les états finaux q de Δ^i sont également finaux dans la reconstruction non simplifié de $\mathcal{E}_{\Delta_i}$. Ainsi, il existe au moins une trajectoire passant par q et q n'est donc pas supprimé par simplification. Donc, q est un état final de Δ_i. □

Ce résultat montre qu'étant données les informations à la date t_i, il n'est pas nécessaire de reconstruire le diagnostic pour connaître les états possibles du système.

3.3.4 Raffinements supplémentaires

Le diagnostic incrémental permet d'obtenir une chaîne de diagnostics I-raffinée. On peut se demander s'il est intéressant de calculer la chaîne de diagnostics raffinée.

3.3.4.1 Synchronisation raffinée

Définition 3.21 (F-restriction)**.**
Soit $A = (Q, E, T, I, F)$ un automate. La F-restriction *de A sur F' notée $A(F')$ est l'automate simplifié de $(Q, E, T, I, F \cap F')$.*

La F-restriction est l'opération symétrique de la I-restriction.

Définition 3.22 (Synchronisation raffinée)**.**
Soit M un automate et $\mathcal{E}_A$ une chaîne d'automates. Soit $(A^1_\odot, \ldots, A^n_\odot)$ la synchronisation incrémentale de M et de $\mathcal{E}_A$ où $\forall i$, $A^i_\odot = (Q^i_\odot, E^i_\odot, T^i_\odot, I^i_\odot, F^i_\odot)$. La synchronisation raffinée *de M et de $\mathcal{E}_A$, notée $M \circledcirc \mathcal{E}_A$ est la séquence d'automates $(A^1_\circledcirc, \ldots, A^n_\circledcirc)$ définie par :*

- *$A^n_\circledcirc = A^n_\odot$,*
- *$\forall i \in \{1, \ldots, n-1\}$, $A^i_\circledcirc$ est la simplification de $A^i_\odot(I^{i+1}_\circledcirc)$ où $I^{i+1}_\circledcirc$ est l'ensemble des états initiaux de $A^{i+1}_\circledcirc$.*

Cette synchronisation consiste à restreindre l'ensemble des états finaux de $A^i_\circledcirc$ sur l'ensemble des états initiaux de $A^{i+1}_\circledcirc$. Il y a donc deux calculs incrémentaux : le premier de l'automate A^1 à A^n pour calculer la synchronisation incrémentale, et le second de A^n à A^1 pour calculer la synchronisation raffinée. Ces deux calculs correspondent aux deux boucles *prévision-postdiction* de [Lar00].

Théorème 3.7.
Soit M un automate et $\mathcal{E}_A$ une chaîne d'automates. Alors, $M \circledcirc \mathcal{E}_A$ est la chaîne d'automates raffinée obtenue par F-raffinements successifs de $M \odot \mathcal{E}_A$.

Démonstration. Un F-raffinement supprime un état de la liste des états finaux d'un morceau d'automate lorsque cet état n'appartient pas à l'ensemble des états initiaux du morceau d'automate suivant. Ainsi, l'opération consistant à remplacer F^i par $I^{i+1} \subseteq F^i$ est équivalent à l'application successive de F-raffinement. Ainsi, $M \circledcirc \mathcal{E}_A$ peut être obtenu par F-raffinements successifs de $M \odot \mathcal{E}_A$.

Nous prouvons maintenant que la chaîne d'automates $M \circledcirc \mathcal{E}_A$ est raffinée. Soit $\mathcal{E}_A$ une chaîne I-raffinée et $\mathcal{E}_{A'}$ la chaîne obtenue après un F-raffinement de $\mathcal{E}_A$ qui a conduit à la suppression de l'état q de l'ensemble des états finaux de l'automate A^i. Alors, pour tout j tel que $j \neq i$ et $j \neq i+1$, $I'^{j+1} = I^{j+1} \subseteq F^j = F'^j$. De plus, $I'^i \subseteq I^i \subseteq F^{i-1} = F'^{i-1}$. Enfin, $I'^{i+1} = I^{i+1} \subseteq F^i$; or on a supprimé de F^i un élément qui n'appartient pas à I^{i+1} ; donc $I'^{i+1} \subseteq F'^i$. Donc, $\mathcal{E}_{A'}$ est une chaîne I-raffinée. Donc, $M \circledcirc \mathcal{E}_A$ est une chaîne I-raffinée.

Par ailleurs, $\forall i$, $A^i_\circledcirc$ est obtenu par simplification de $(Q^i_\odot, E^i_\odot, T^i_\odot, I^i_\odot, I^{i+1}_\circledcirc)$. Donc, $F^i_\circledcirc \subseteq I^{i+1}_\circledcirc$. Donc, $M \circledcirc \mathcal{E}_A$ est une chaîne F-raffinée. Et donc, $M \circledcirc \mathcal{E}_A$ est une chaîne raffinée.

□

3.3.4.2 Calcul non incrémental du diagnostic

Définition 3.23 (Diagnostic raffiné)**.**
Soit Mod l'automate représentant le modèle du système. Soit Obs l'automate des obser-

vations et $\mathcal{E}_{Obs}$ un découpage correct de Obs. Le diagnostic raffiné *du système est défini comme suit : $\mathcal{E}_\Delta = Mod \circledcirc \mathcal{E}_{Obs}$.*

Résultat 3.6.
Le diagnostic raffiné $\mathcal{E}_\Delta$ est un découpage correct du diagnostic Δ.

Démonstration. Ce résultat est une conséquence logique du théorème 3.7, du résultat 3.3 et du théorème 3.5. □

Le diagnostic raffiné produit une chaîne d'automates raffinée, c'est-à-dire ne comportant aucun état inutile. Ce résultat est très intéressant puisqu'il signifie que la chaîne de diagnostics est pratiquement identique au diagnostic reconstruit.

3.3.4.3 Calcul incrémental du diagnostic

Soit $Obs_1, \ldots, Obs_{i+1}$, $i+1$ automates tels que $\forall k \in \{1, \ldots, i\}$, Obs_k est un préfixe de Obs_{k+1}. Considérons qu'il existe $Obs^1, \ldots, Obs^{i+1}$ tel que $\forall k \in \{1, \ldots, i+1\}$, $(Obs^1, \ldots, Obs^k)$ est un découpage correct de Obs_k.

On note $\mathcal{E}_{\Delta_i} = (\Delta^1, \ldots, \Delta^i)$ le diagnostic par morceaux de Obs_i. $\mathcal{E}_{\Delta_i}$ est un découpage correct de $\Delta_i = Mod \otimes Obs_i$. On note $\mathcal{E}_{\Delta_{i+1}}$ le diagnostic par morceaux de Obs_{i+1}. Alors, on a le résultat suivant :

Résultat 3.7.
$\mathcal{E}_{\Delta_{i+1}} = (\Delta'^1, \ldots, \Delta'^i, \Delta^{i+1})$ *où* $\Delta^{i+1} = (Mod^\# \otimes Obs^{i+1})[F^i_\Delta]$ *et* $\forall k \in \{1, \ldots, i\}$, $\Delta'^k = \Delta^k(I^{k+1}_{\Delta'})$ *avec* $I^{k+1}_{\Delta'}$ *est l'ensemble d'états finaux de* Δ'^{k+1}.

On voit que le calcul incrémental par diagnostic raffiné nécessite de recalculer par F-raffinements les automates Δ^k. Nous avons vu que la complexité d'un diagnostic incrémental ne devait dépendre que de la taille de Obs^{i+1}. Ici, puisqu'il est nécessaire de F-raffiner les précédents automates, la complexité dépend également de la taille de Obs. Ce n'est donc pas une solution acceptable pour le calcul incrémental du diagnostic.

Aussi, par la suite nous ne considérons pas le diagnostic raffiné du diagnostic. On pourrait cependant s'intéresser à un diagnostic *partiellement* raffiné, c'est-à-dire qui consiste à effectuer les F-raffinements uniquement pour les k derniers morceaux d'automates (k petit, synchronisation k-raffinée).

3.4 Découpage des observations

Nous avons jusqu'ici considéré que la chaîne des automates des observations était déjà calculée et nous avons présenté l'opération (reconstruction) permettant de retrouver l'automate des observations. Nous présentons à présent le découpage de l'automate des observations. Nous donnons tout d'abord dans cette section une définition de découpage plus stricte appelée *découpage temporel*. Nous présentons ensuite une technique permettant de découper de manière générale un automate pour obtenir une chaîne d'automates, et nous appliquons ce résultat dans le cadre hors-ligne à la construction de la chaîne d'automates des observations.

3.4.1 Découpage temporel

Le découpage temporel consiste à considérer les morceaux d'automates comme les comportements possibles ayant eu lieu pendant une période donnée. Ces périodes sont appelées *fenêtres temporelles*. À chaque automate correspond alors une fenêtre temporelle.

Définition 3.24 (Séquence correcte de fenêtres temporelles)**.**
Soit $t_0, \ldots, t_n$ des instants et $[t_0, t_n]$ la fenêtre temporelle globale du diagnostic. Une séquence de fenêtres temporelles est correcte *respectivement à $[t_0, t_n]$ s'il s'agit d'une séquence $\mathcal{W} = (\mathcal{W}_1, \ldots, \mathcal{W}_i, \ldots, \mathcal{W}_n)$ telle que $\mathcal{W}_1 = [t_0, t_1]$, $\mathcal{W}_n = [t_{n-1}, t_n]$, et $\mathcal{W}_i = [t_{i-1}, t_i]$.*

La fenêtre temporelle globale $[t_0, t_n]$ est découpée en une séquence $\mathcal{W}$ de fenêtres temporelles plus courtes. Il faut à présent définir le découpage temporel.

Définition 3.25 (Découpage temporellement correct)**.**
Soit Obs_n l'automate des observations sur la fenêtre temporelle $[t_0, t_n]$. La chaîne d'automate $\mathcal{E}_{Obs_n} = (Obs^{\mathcal{W}_1}, \ldots, Obs^{\mathcal{W}_n})$ est temporellement correcte *par rapport à $\mathcal{W} = (\mathcal{W}_1, \ldots, \mathcal{W}_i, \ldots, \mathcal{W}_n)$ si :*

- *le découpage est correct,*
- *$\mathcal{W}$ est une séquence correcte de fenêtres temporelles respectivement à $[t_0, t_n]$, et*
- *$\forall$chem trajectoire de $Obs^{\mathcal{W}_i}$, les transitions ont eu lieu durant $[t_{i-1}, t_i]$ (c'est-à-dire que les observations ont été émises par le système durant la période $\mathcal{W}_i$).*

Le découpage temporellement correct consiste donc à créer une chaîne d'automates dont chaque morceau corresponde à l'activité durant une période donnée. Considérons la reconstruction Obs_n de $\mathcal{E}_{Obs_n}$. Alors, F^i, l'ensemble des états finaux de $Obs^{\mathcal{W}_1}$ est l'ensemble des états sur lesquels on peut se situer à la date t_i.

Que se passe-t-il lorsqu'on ne sait pas si une observation a été émise avant ou après une date t_i ? Prenons le cas de l'observation a et considérons pour simplifier qu'aucune autre observation n'a été envoyée pendant l'intervalle de temps. Le découpage temporellement correct est donné sur la figure 3.8. La transition étiquetée par a dans le premier automate représente le fait que l'observation a été émise avant t_i, alors que celle du second automate représente l'émission après t_i. Cependant, une seule transition peut être franchie. Donc, ce découpage représente le fait que l'observation a pu être émise avant ou après t_i.

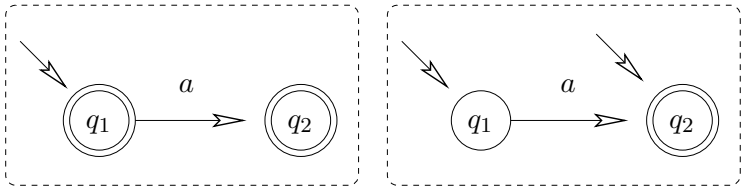

FIG. 3.8 – Exemple de découpage temporellement correct

Les résultats de la section 3.3 peuvent être utilisés dans le cadre d'un découpage temporellement correct. Notons $\forall i, \mathcal{E}_{Obs_{\mathcal{W}_i}} = (Obs^{\mathcal{W}_1}, \ldots, Obs^{\mathcal{W}_i})$. Soit $i < n$, et $\mathcal{E}_{\Delta_{\mathcal{W}_i}} = \mathcal{E}_{Obs_{\mathcal{W}_i}} \odot Mod = (\Delta^{\mathcal{W}_1}, \ldots, \Delta^{\mathcal{W}_i})$. Alors, $\mathcal{E}_{\Delta_{\mathcal{W}_{i+1}}} = \mathcal{E}_{Obs_{\mathcal{W}_{i+1}}} \odot Mod$ peut être calculé comme suit :

Résultat 3.8.
$\mathcal{E}_{\Delta_{\mathcal{W}_{i+1}}} = (\Delta^{\mathcal{W}_1}, \ldots, \Delta^{\mathcal{W}_i}, \Delta^{\mathcal{W}_{i+1}})$ *avec* $\Delta^{\mathcal{W}_{i+1}} = (Obs^{i+1} \otimes Mod^{\#})[F_{\Delta}^{\mathcal{W}_i}]$ *où* $F_{\Delta}^{\mathcal{W}_i}$ *est l'ensemble des états finaux de* $\Delta^{\mathcal{W}_i}$.

Le découpage temporel pour le diagnostic a de nombreux intérêts. Tout d'abord, le diagnostic fournit l'ensemble des états possibles du système à la date t_i. De plus, il est possible de s'intéresser de manière précise à une période donnée, alors que le découpage était considéré comme quelconque jusque là. Un autre point intéressant concerne le découpage en-ligne. En effet, il apparait plus facile de considérer une date donnée pour découper l'automate, plutôt que d'effectuer un découpage quelconque.

3.4.2 Découpage d'un automate

Dans le cadre hors-ligne, le découpage se fait simplement en déterminant l'ensemble des états *frontière* appelé *ensemble de découpage*.

Définition 3.26 (Ensemble correct de découpage)**.**
Soit $A = (Q, E, T, I, F)$ *un automate et* Q' *un ensemble d'états.* Q' *est appelé un* ensemble correct de découpage *si :*

- *toute trajectoire* chem *de* A *passe par un état* $q \in Q'$,
- $\forall$chem $= ((q_0, \ldots, q_m), (l_1, \ldots, l_m))$ *trajectoire de* A, $\forall i, k, j$ *tels que* $i < k < j$, *alors* $q_i \in Q' \wedge q_j \in Q' \Rightarrow q_k \in Q'$.

Un ensemble correct de découpage se définit comme un sous-ensemble de Q qui permet de partitionner les états $Q \setminus Q'$ en un ensemble d'états qui sont atteints *avant* les états de Q' et un ensemble d'états qui sont atteints *après* les états de Q' sur les trajectoires. La figure 3.9 donne une représentation symbolique de cette définition. Considérons que le rectangle représente l'ensemble des trajectoires possibles (représentées par les flèches) sur un automate. Alors, Q_1 n'est pas un ensemble correct de découpage de l'automate puisque certaines trajectoires ne passent pas par un état de Q_1. D'autre part, Q_2 n'est pas un ensemble correct de découpage puisqu'il existe un ensemble d'états (au centre de Q_2) qui ne se situent ni avant ni après Q_2 (Q_2 ne respecte donc pas la seconde condition de la définition). En revanche, Q_3 est un ensemble correct de découpage.

Définition 3.27 (Découpage d'un automate)**.**
Soit $A = (Q, E, T, I, F)$ *un automate et* Q' *un ensemble d'états. Le* découpage *de* A *par* Q' *est la séquence d'automates* (A^1, A^2) *définie par :*

- A^1 *est l'automate simplifié de* (Q, E, T, I, Q') *et*
- A^2 *est l'automate simplifié de* (Q, E, T, Q', F).

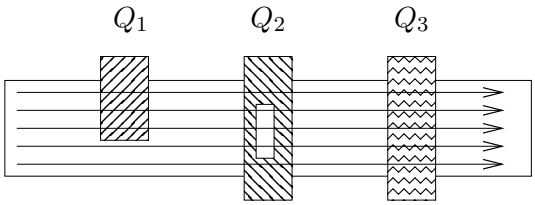

FIG. 3.9 – Exemple d'ensembles de découpage

Théorème 3.8.
Soit $A = (Q, E, T, I, F)$ un automate et Q' un ensemble correct de découpage. Soit $\mathcal{E}_A = (A^1, A^2)$ le découpage de A par Q'. Alors, $\mathcal{E}_A$ est une chaîne d'automates et $\mathcal{E}_A$ est un découpage correct de A.

Démonstration. On note $A^i = (Q^i, E^i, T^i, I^i, F^i)$.

Tout d'abord, montrons que la séquence d'automates est une chaîne.

- $E^1 = E^2$,
- Soit $q \in Q^1 \cap Q^2$. Alors, il existe une trajectoire de $q_0^1 \in I^1$ à $q_f^1 \in F^1$ passant par q sur A^1, et une trajectoire de $q_0^2 \in I^2$ à $q_f^2 \in F^2$ passant par q sur A^2. Ainsi, il existe un chemin entre q_f^1 et q_0^2 passant par q sur A. Or $q_f^1 \in F^1 \subseteq Q'$ et $q_0^2 \in I^2 \subseteq Q'$. Donc, d'après la seconde condition de la définition d'un ensemble correct de découpage, $q \in Q'$, et donc $q \in F^1 \cap I^2$.
- $\forall q, q'$, si $\{q, q'\} \subseteq Q^1 \cap Q^2$ alors $\forall p$, chemin de A^1 entre q et q', p est aussi un chemin sur A^2 (et réciproquement) ? Cette relation est respectée puisque les transitions de A^1 et A^2 sont les transitions de A entre les états de A^1 et A^2.

La chaîne $\mathcal{E}_A$ est-elle un découpage correct de A ? Soit chem une trajectoire sur $\mathcal{E}_A$. Alors, puisque les transitions de $\mathcal{E}_A$ proviennent de A, chem est également une trajectoire de A. Soit chem $= ((q_0, \ldots, q_m), (l_1, \ldots, l_m))$ une trajectoire de A. Alors, il existe i tel que $q_i \in Q'$ (première condition de la définition d'ensemble correct de découpage). Donc, $((q_0, \ldots, q_i), (l_1, \ldots, l_i))$ est une trajectoire de A^1 et $((q_i, \ldots, q_m), (l_{i+1}, \ldots, l_m))$ est une trajectoire de A^2. Donc, chem est une trajectoire de $\mathcal{E}_A$. Ainsi, les ensembles de trajectoires de A et $\mathcal{E}_A$ sont identiques. □

Nous avons donc montré qu'il était possible de découper un automate de manière générale. Nous montrons comment utiliser ces résultats pour le cas particulier du découpage de l'automate des observations.

3.4.3 Application aux observations

Nous reprenons ici les hypothèses formulées en section 2.4. Nous appliquons les résultats présentés précédemment pour découper l'automate des observations hors-ligne, c'est-à-dire une fois obtenu l'ensemble des observations reçues par le superviseur. Nous présentons d'abord un découpage non temporel, puis nous étendons le résultat à un

découpage temporel. Nous considérons ici que les automates des observations ont été construits par ensembles possibles (voir le résultat 2.3 page 59).

3.4.3.1 Découpage correct non temporel

Nous cherchons à présent un ensemble correct de découpage. Nous proposons ici une solution possible.

Définition 3.28 (Fermeture possible)**.**
Soit $\mathcal{O}$ l'ensemble des observations émises et $\prec$ la relation d'ordre partiel d'émission des observations. Soit O et O' deux ensembles d'observations. On dit que O' est une fermeture possible *de O, noté $\blacktriangleleft(O)$, si :*
- $\lhd(O')$,
- $O' \supseteq O$, *et*
- $\exists o \in O, \nexists o' \in O', o \prec o'$.

Une fermeture possible d'un ensemble O d'observations est un sur-ensemble possible O' de O tel qu'une observation o de O peut être la dernière observation de O' à avoir été émise.

Exemple : Reprenons l'exemple de la figure 2.10, page 60. Considérons l'ensemble $O = \{A\}$. Il existe deux fermetures possibles de O. La première fermeture possible est $\{A\}$ et la seconde est $\{A, B\}$. En revanche, $\{A, B, C\}$ n'est pas une fermeture possible de O car l'observation C est postérieure à toute observation de O.

Théorème 3.9.
Soit $\mathcal{O}$ l'ensemble des observations émises et $\prec$ la relation d'ordre partiel d'émission des observations. Soit Obs l'automate des observations obtenu par le résultat 2.3. Soit $O \subseteq \mathcal{O}$ un ensemble d'observations et Q' l'ensemble des fermetures possibles de O. Alors, Q' est un ensemble correct de découpage de Obs.

Démonstration. Nous démontrons les deux conditions de la définition d'ensemble correct de découpage.
- Toute trajectoire chem de *Obs* passe par un état de Q'. Soit chem $= ((q_0, \ldots, q_m), (l_0, \ldots, l_m))$. Puisque $q_0 = \emptyset$ et $q_m = \mathcal{O}$, il existe i tel que $O \nsubseteq q_{i-1}$ et $O \subseteq q_i$. Nous savons que $\lhd(q_i)$. On note q tel que $q_i = q_{i-1} \uplus q$. Il existe $o \in O$ tel que $o \in q$ (par définition de i et de q). Ainsi, d'après la définition de l'automate (résultat résultat 2.3), $\nexists o' \in q,\ o \prec o'$. De plus, $\nexists o' \in q_{i-1},\ o \prec o'$ (car $\lhd(q_{i-1})$ et $o \notin q_{i-1}$). Donc, $\nexists o' \in q_i = q_{i-1} \uplus q,\ o \prec o'$. Nous avons prouvé que q_i est une fermeture possible de O. Donc, chem passe bien par Q'.
- $\forall$chem $= ((q_0, \ldots, q_m), (l_1, \ldots, l_m)), \forall i, k, j$ tels que $i < k < j$ et $q_i \in Q' \wedge q_j \in Q'$. Nous allons prouver que $q_k \in Q'$. Il faut donc prouver les trois conditions de la définition 3.26. Remarquons que $q_k \in Q$, et donc d'après le résultat 2.3, $\lhd(q_k)$ (1).

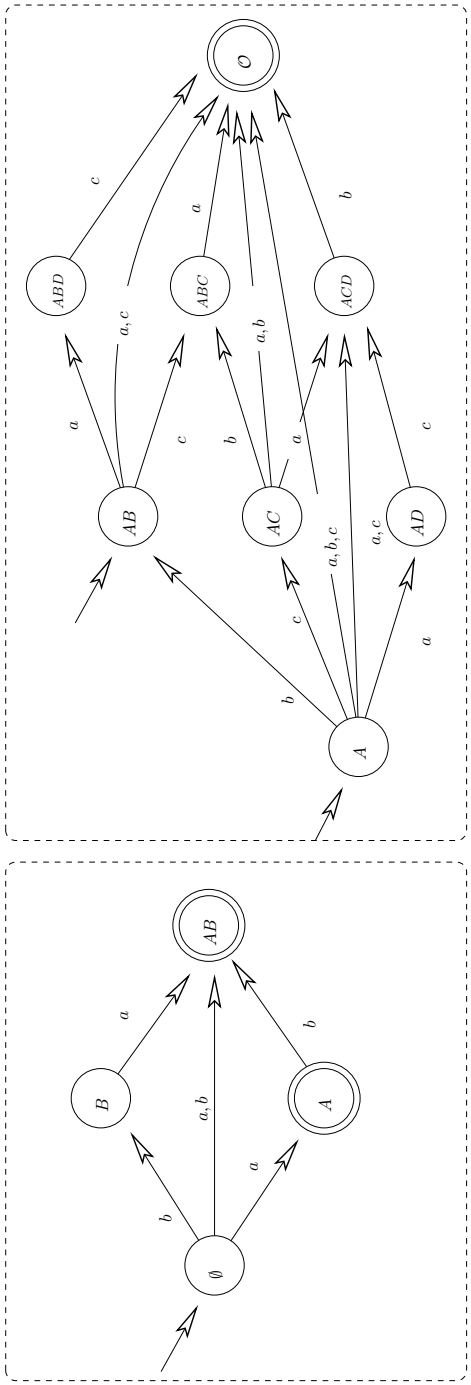

FIG. 3.10 – Automate des observations de la figure 2.10 découpé par l'ensemble des fermetures possibles de $\{A\}$

L'automate défini dans le résultat 2.3 indique qu'il n'existe une transition entre q et q' que si $q \subseteq q'$ (en considérant les transitions implicites, on peut avoir $q = q'$). Donc, on a $O \subseteq q_i \subseteq q_k \subseteq q_j$ (2). Enfin, puisque $\exists o \in O$, $\nexists o' \in q_j$, $o \prec o'$, et puisque $O \subseteq q_k \subseteq q_j$, alors cette condition est également vérifiée pour q_k (3).

Donc, Q' est bien un ensemble de découpage. □

Exemple : Nous découpons l'automate de la figure 2.10 avec l'ensemble des fermetures possibles de $O = \{A\}$. Le résultat est donné sur la figure 3.10, page 87.

3.4.3.2 Découpage temporel

Considérons à présent le découpage temporel. Soit o une observation émise à la date t_i et $O = \{o\}$. Alors, le découpage $(Obs^{\mathcal{W}_1}, Obs^{\mathcal{W}_2})$ de l'automate des observations par l'ensemble des fermetures possibles de O est temporellement correct pour $\mathcal{W} = \{\mathcal{W}_1, \mathcal{W}_2\}$ avec $\mathcal{W}_1 = [t_0, t_i]$ et $\mathcal{W}_2 = [t_i, t_n]$. En effet, l'ensemble Q' des fermetures possibles pour O est l'ensemble des ensembles d'observations ayant pu être émis avant t_i. Ainsi, les transitions représentant l'émission d'une observation $o' \prec o$ sont présentes dans l'automate $Obs^{\mathcal{W}_1}$ (puisque $\forall q \in Q'$, $o' \in q$ et donc $\forall q \in Q^{\mathcal{W}_1}$, $o' \in q$). De même, les observations o' émises après o ($o \prec o'$) sont présentes dans l'automate $Obs^{\mathcal{W}_2}$. En revanche, les observations o' telles que $o' \not\prec o$ et $o \not\prec o'$ sont présentes sur les deux automates $Obs^{\mathcal{W}_1}$ et $Obs^{\mathcal{W}_2}$.

Considérons à présent qu'on souhaite découper l'automate à la date t_i et qu'il n'y a aucune observation émise à cette date. Alors, on peut considérer qu'il existe une observation fictive o_{t_i} à la date t_i et étendre la définition de fermeture possible à une date. On considère que la relation d'ordre partiel $\prec$ peut être étendue à une relation $\prec'$ telle que :

- $o \prec' o'$ si $o \prec o'$,
- $o \prec' t_i$ si l'observation o a été émise avant t_i,
- $t_i \prec' o$ si l'observation o a été émise après t_i.

Définition 3.29 (Fermeture possible à une date)**.**
Soit $\mathcal{O}$ un ensemble d'observations reçues et $\prec$ une relation d'ordre partiel d'émission des observations. Soit t_i une date et $\prec'$ l'extension partielle de $\prec$ pour t_i. Soit $O \subseteq \mathcal{O}$ un ensemble d'observations. On dit que O est une fermeture possible *de t_i, noté $\blacktriangleleft(t_i)$, si :*

- $\lhd(O)$,
- $\forall o \in \mathcal{O}$, $o \prec' t_i \Rightarrow o \in O$, *et*
- $\forall o \in \mathcal{O}$, $t_i \prec' o \Rightarrow o \notin O$.

L'automate des observations peut alors être temporellement correctement découpé par l'ensemble des fermetures possibles de t_i.

3.5 Conclusion

Dans cette partie, nous avons défini le calcul incrémental du diagnostic. Nous avons introduit le formalisme de chaîne d'automates pour le diagnostic incrémental. Une chaîne d'automates est une représentation *par morceaux* d'un automate. L'automate des observations est ainsi découpé en une chaîne d'automates. Le diagnostic est effectué sur une première partie de la chaîne, puis incrémentalement en ajoutant un morceau d'automate à la chaîne. Il est possible d'obtenir l'automate d'origine par l'opération de reconstruction.

Puisque le diagnostic est défini comme la synchronisation de l'automate du modèle et de l'automate des observations, et puisque l'automate des observations est découpé en une chaîne d'automates, nous avons défini plusieurs synchronisations possibles d'un automate et d'une chaîne d'automates (synchronisation par morceau, synchronisation incrémentale, synchronisation raffinée, synchronisation k-raffinée). Nous avons discuté des avantages et inconvénients de ces différentes synchronisations dans un contexte de diagnostic incrémental.

Enfin, nous avons présenté comment découper correctement un automate et quelles sont les propriétés à vérifier. Nous avons ensuite appliqué ces résultats dans le contexte des hypothèses effectuées sur les observations au chapitre précédent.

Chapitre 4

Calcul en-ligne du diagnostic par chaînes d'automates

Le chapitre précédent a présenté une technique pour calculer incrémentalement le diagnostic. Nous étendons ces résultats pour permettre le calcul en-ligne du diagnostic.

Le diagnostic en-ligne consiste à calculer un diagnostic du système pendant que celui-ci est en fonctionnement. Lorsque le diagnostic est effectué à plusieurs dates, il doit être effectué de manière incrémentale pour bénéficier des calculs précédents. Dans le cas contraire, la complexité augmenterait à chaque nouveau calcul puisque le nombre d'observations augmente.

Dans notre contexte de calcul incrémental du diagnostic par chaînes d'automates, la difficulté est de construire la chaîne des observations (découpage correct de l'automate des observations) alors que l'automate des observations n'est pas complètement connu.

Le chapitre est divisé comme suit. Dans un premier temps, le diagnostic en-ligne est formellement défini. Ensuite, nous montrons comment résoudre ce problème par chaîne d'automates, et nous définissons notamment le découpage en-ligne. Enfin, nous montrons une technique pour découper les observations selon certaines hypothèses que nous posons.

4.1 Définition du diagnostic en-ligne

Le terme correct qu'il conviendrait d'employer est le calcul en-ligne du diagnostic, mais pour simplifier, nous employons le terme de diagnostic en-ligne.

Le diagnostic en-ligne est le problème suivant : étant donné un flux d'observations sur le système, il s'agit d'effectuer à différentes dates un diagnostic du système (voir définition 1.9, page 27). Notons que si un diagnostic est effectué à une date t, il ne cherche pas forcément à expliquer le comportement jusqu'à la date t mais peut vouloir expliquer le comportement jusqu'à une date t' (généralement $t' < t$). Le but du diagnostic en-ligne est de surveiller le comportement du système pour pouvoir intervenir en cas de dysfonctionnement. Le diagnostic doit trouver les comportements qui ont eu lieu dans le système alors que toutes les observations émises ne sont pas parvenues au superviseur.

Nous considérons qu'il existe une date t_n telle qu'aucune observation ne peut être reçue après cette date. On note O^* l'ensemble des observations reçues à la date t_n des canaux de communication entre le système et le superviseur. On note Obs^* l'automate des observations construit à partir de cet ensemble.

Définition 4.1 (Diagnostic en-ligne par automate)**.**
Soit Obs_i un automate préfixe de Obs^*. Le diagnostic en-ligne Δ_i du système à la date t_i est l'automate : $\Delta_i = Mod \otimes Obs_i$.

Le diagnostic en-ligne consiste donc à effectuer un diagnostic avec un préfixe de l'automate des observations Obs^*.

Remarquons que le nombre d'observations reçues par le système est croissant. Ainsi, l'automate Obs_i est de plus en plus proche de Obs^* à mesure qu'on reçoit de nouvelles observations : $\forall i$, Obs_i est préfixe de Obs_{i+1}.

Le diagnostic en-ligne consiste à effectuer à une date $t < t_n$ le diagnostic du système et suivre le comportement du système. Rappelons que, comme nous l'avons vu en 1.3.1.1, le diagnostic en-ligne à la date t ne calcule pas forcément les comportements du système jusqu'à la date t. Ainsi, dans certains cas, on n'a pas reçu suffisamment d'observations (parce que les observations émises n'ont pas été encore transmises) sur le comportement actuel. Dans ces cas, il est impossible de savoir ce qui s'est passé sur le système. Dans le pire des cas, n'importe quelle panne peut avoir eu lieu. Aussi, le calcul du diagnostic à la date actuelle n'est pas pertinent, et on se contente de calculer les comportements jusqu'à une date $t' < t$.

Nous avons noté deux difficultés principales du diagnostic en-ligne :

D'une part, puisque le superviseur doit fournir régulièrement des diagnostics, il faut que le calcul soit efficace et rapide pour que le calcul soit terminé avant qu'un nouveau diagnostic soit demandé. Or, par définition, le nombre d'observations reçues par le superviseur croît linéairement. Ainsi, la complexité augmente régulièrement. Aussi, il est nécessaire d'adopter une approche incrémentale et d'utiliser les résultats du chapitre précédent. La difficulté est alors d'être capable de découper l'automate des observations alors qu'il n'est pas complet.

D'autre part, puisque le système est en fonctionnement, il génère régulièrement des observations qui sont transmises par les canaux de communication. Cependant, dans la plupart des cas réels, ces canaux induisent des délais de transmission. Ainsi, par rapport à un diagnostic hors-ligne, on a des incertitudes supplémentaires sur les observations émises par le système.

4.2 Calcul du diagnostic en-ligne

Nous présentons maintenant comment nous définissons le diagnostic en-ligne pour le diagnostic défini comme la synchronisation entre l'automate modélisant le comportement du système et l'automate des observations. Nous nous appuyons notamment sur le formalisme des chaînes d'automates et les résultats présentés au chapitre précédent.

Pour effectuer le diagnostic en-ligne, il nous faut définir le découpage en-ligne. Nous appliquons ensuite ce résultat au diagnostic en-ligne. Enfin, nous montrons comment considérer un découpage temporel pour le cadre en-ligne.

4.2.1 Découpage en-ligne des observations

Pour permettre un calcul incrémental, on effectue un diagnostic sur une chaîne d'automates *partiellement découpée*, c'est-à-dire une chaîne ne comprenant que les premiers automates du découpage correct.

Les définitions données dans ce paragraphe sont générales et ne s'appliquent pas nécessairement au contexte de diagnostic.

Définition 4.2 (Découpage partiel d'un automate)**.**
Soit Obs un automate. On dit que $\mathcal{E}_{Obs} = (Obs^1, \ldots, Obs^i)$ *est un* découpage partiel *de Obs si il existe* Obs' *tel que* $(Obs^1, \ldots, Obs^i, Obs')$ *est un découpage correct de Obs.*

Considérons $(Obs^1, \ldots, Obs^n)$ un découpage correct de Obs^*. Le diagnostic en-ligne va consister à effectuer un diagnostic sur un découpage partiel $(Obs^1, \ldots, Obs^i)$ $(i < n)$. Ainsi, ce calcul pourra être réutilisé pour un calcul incrémental du diagnostic. La difficulté repose sur le fait que l'on ne connaît pas Obs^* à la date t.

On considère qu'on se situe à la date t. On dispose de certaines informations sur les observations émises par le système. Ces informations nous permettent de définir l'ensemble OBS des automates des observations émises qui pourront être construits à la date t_n. Remarquons que $Obs^* \in OBS$. D'autre part, l'ensemble OBS construit à une date $t' > t$ est inclus dans OBS. En fait le nombre d'automates des observations de OBS diminue régulièrement jusqu'à ce qu'il ne comporte plus que le seul élément Obs^* à la date t_n.

Nous donnons donc une définition de découpage *en-ligne*, qui est une opération consistant à découper partiellement un ensemble d'automates.

Définition 4.3 (Découpage correct en-ligne)**.**
Soit OBS un ensemble d'automates. La chaîne d'automates $\mathcal{E}_{OBS} = (Obs^1, \ldots, Obs^i)$ *est un* découpage correct en-ligne *de OBS si* $\forall Obs \in OBS$, $\mathcal{E}_{OBS}$ *est un découpage partiel de Obs.*

Cette définition indique que quelles que soient les observations reçues par la suite $(\forall Obs \in OBS)$, il sera possible d'ajouter un automate Obs' au découpage correct en-ligne pour que la nouvelle chaîne ainsi formée soit un découpage correct de Obs. Ainsi, il est possible de raisonner sur le début de la chaîne et de commencer le diagnostic à l'aide du découpage en-ligne.

La figure 4.1 donne un exemple de découpage correct en-ligne. Les trois automates en haut de la figure sont les automates de OBS. Généralement, l'ensemble des automates est infini puisque le système peut *a priori* émettre autant d'observations qu'il le veut. La chaîne d'automates en bas de la figure est le découpage en-ligne. On voit ainsi que si l'automate final Obs^* est le troisième automate de l'ensemble, alors il sera possible

d'ajouter à la chaîne d'automates un automate comportant les états 3 (initial), 5 et 6 (final) ainsi que les transitions entre ces états.

Il existe plusieurs manières de découper en-ligne un ensemble d'automates. Ainsi, l'ensemble *OBS* présenté à la figure 4.1 peut être découpé en-ligne par la chaîne d'automates ne comportant que le premier automate de la chaîne présentée sur la même figure. On voit cependant qu'il est plus intéressant d'avoir le découpage présenté sur la figure, puisqu'il est possible de raisonner sur un morceau plus important de l'automate final. Il est ainsi possible de définir une relation d'ordre partiel sur les découpages en-ligne.

Définition 4.4 (Précision d'un découpage en-ligne)**.**
Soit OBS un ensemble d'automates. Soit $\mathcal{E}_{OBS}$ *et* $\mathcal{E}'_{OBS}$ *deux découpages corrects en-ligne de OBS. On dit que* $\mathcal{E}_{OBS}$ *est un découpage en-ligne* plus précis *que* $\mathcal{E}'_{OBS}$*, noté* $\mathcal{E}'_{OBS} \preceq \mathcal{E}_{OBS}$*, si* $\forall$chem*, trajectoire de* $\mathcal{E}_{OBS}$*, il existe une trajectoire de* $\mathcal{E}'_{OBS}$ *préfixe de* chem*.*

Un découpage en-ligne plus précis comporte des trajectoires plus *avancées*. Notons qu'on peut avoir deux chaînes d'automates $\mathcal{E}_{OBS}$ et $\mathcal{E}'_{OBS}$ représentant des ensembles de trajectoires différents tels que $\mathcal{E}_{OBS} \preceq \mathcal{E}'_{OBS}$ et $\mathcal{E}'_{OBS} \preceq \mathcal{E}_{OBS}$.

Le but du découpage en-ligne est de commencer à construire la chaîne d'automates à une date $t < t_n$ pour commencer le calcul du diagnostic immédiatement. Dans le cadre d'un calcul en-ligne du diagnostic, le découpage en-ligne est effectué à différentes dates $t_1, \ldots, t_i, \ldots$ On va alors chercher généralement à ajouter un automate à la chaîne d'automates déjà construite à la date t_{i-1} pour le découpage à la date t_i. Le découpage en-ligne à la date t_{i-1} a été effectué à partir d'un ensemble d'automates OBS_{i-1}. L'ensemble OBS_i d'automates à t_i est tel que $OBS_i \subseteq OBS_{i-1}$.

Définition 4.5 (Découpage incrémental correct en-ligne)**.**
Soit OBS_i *et* OBS_{i-1} *deux ensembles d'automates tels que* $OBS_i \subseteq OBS_{i-1}$*. Soit* $\mathcal{E}_{OBS_{i-1}} = (Obs^1, \ldots, Obs^{i-1})$*, un découpage en-ligne de* OBS_{i-1}*. La chaîne d'automates* $\mathcal{E}_{OBS_i}$ *est un* découpage incrémental correct en-ligne *de* OBS_i *si* $\mathcal{E}_{OBS_i}$ *est un découpage correct en-ligne de* OBS_i *et si* $\exists Obs^i$ *tel que* $\mathcal{E}_{OBS_i} = (Obs^1, \ldots, Obs^{i-1}, Obs^i)$*.*

4.2.2 Application au diagnostic en-ligne

Nous considérons d'abord le diagnostic en-ligne utilisant les chaînes d'automates et nous considérons ensuite le calcul incrémental du diagnostic en-ligne.

4.2.2.1 Diagnostic en-ligne par chaîne d'automates

Soit O l'ensemble des observations reçues à la date t. Soit OBS_i l'ensemble des automates des observations émises à la date t_n étant donné O. Soit $\mathcal{E}_{OBS_i}$ un découpage incrémental correct en-ligne de OBS_i.

Résultat 4.1.
$\Delta_i = Mod \otimes Sli^{-1}(\mathcal{E}_{OBS_i})$ *est un diagnostic en-ligne du système.*

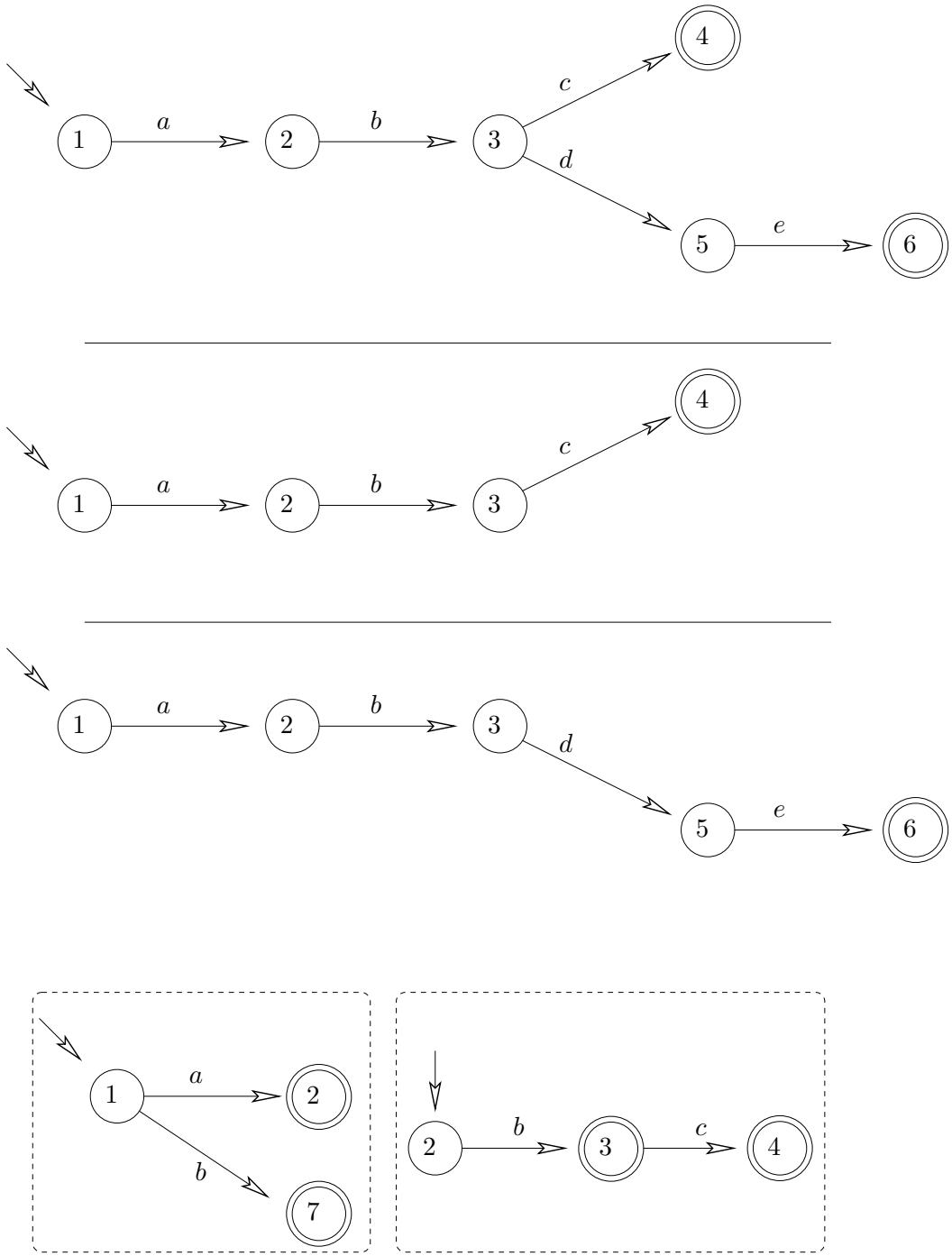

FIG. 4.1 – Exemple de découpage en-ligne

Grâce à ce résultat, il est possible d'utiliser le découpage en-ligne pour calculer le diagnostic en-ligne.

Dans le cas général, notamment dans le cadre du diagnostic en-ligne, nous ne calculons pas le diagnostic global Δ_i. En effet, le système étant censé fonctionner pendant une longue période sans arrêt, la taille du diagnostic augmente régulièrement jusqu'à saturer la mémoire. D'autre part, on peut considérer que dans le cadre du diagnostic en-ligne, seuls les derniers comportements du système (et donc les derniers morceaux de l'automate Δ_i) sont intéressants. Pour ces deux raisons, on considère donc un découpage du diagnostic Δ_i. Le diagnostic est alors calculé par :

$$Mod\ op\ \mathcal{E}_{OBS_i}$$

où *op* est un des opérateurs de synchronisation présentés dans le chapitre 3 ($\otimes$, $\odot$, $\circledcirc$ ou l'opération de synchronisation k-raffinée). Notons que l'opérateur $\otimes$ doit être rejeté puisque les derniers morceaux de la chaîne de diagnostic comportent de nombreux états inutiles.

4.2.2.2 Calcul incrémental du diagnostic en-ligne par chaîne d'automates

Nous avons donc montré comment calculer le diagnostic en-ligne en utilisant les chaînes d'automates. Pour être efficace, ce calcul doit être effectué de manière incrémentale.

Soit $\mathcal{E}_{Obs_{i-1}} = (Obs^1, \ldots, Obs^{i-1})$ un découpage correct en-ligne des observations à la date t_{i-1} et $\mathcal{E}_{Obs_i} = (Obs^1, \ldots, Obs^i)$ un découpage incrémental correct en-ligne à la date t_i. Soit $\mathcal{E}_{\Delta_{i-1}} = (\Delta^1, \ldots, \Delta^{i-1})$ un découpage du diagnostic en-ligne à la date t_{i-1} avec $\forall j \in \{1, \ldots, i-1\}$, $\Delta^j = (Q^j, E^j, T^j, I^j, F^j)$.

Résultat 4.2.
Un découpage $\mathcal{E}_{\Delta_i}$ *du diagnostic en-ligne est* $(\Delta^1, \ldots, \Delta^{i-1}, \Delta^i)$ *où* $\Delta^i = (Mod^- \otimes Obs^i)[F^{i-1}]$.

On voit avec ce résultat qu'il suffit de calculer $\Delta^i = (Mod^- \otimes Obs^i)[F^{i-1}]$ pour obtenir le diagnostic en-ligne du système. Rappelons la propriété suivante : dans la chaîne $\mathcal{E}_{\Delta_i}$, le morceau Δ^i ne comporte pas d'état inutile.

Nous nous intéressons maintenant au découpage temporel des observations.

4.2.3 Découpage temporel en-ligne des observations

Nous ne considérons pas ici le caractère incrémental du calcul, et nous considérons donc que le diagnostic en-ligne se fait à une date t.

Nous avons vu qu'il était possible de considérer différents automates des observations Obs_i, la seule contrainte étant que Obs_{i-1} soit préfixe de Obs_i. Nous proposons à présent de faire ce découpage de manière temporelle. Ainsi, il est possible de raisonner sur des périodes de temps et de considérer le comportement du système pendant une période donnée. Par ailleurs, se forcer à effectuer un découpage temporel permet de s'assurer de donner régulièrement plus de précision à la chaîne d'automates. En effet,

une chaîne d'automates ne comportant qu'un automate constitué d'un seul état à la fois initial et final, est un découpage correct d'un automate ayant un seul état initial. Cependant, ce découpage est totalement imprécis. Ici, nous nous forçons à construire le morceau d'automate des observations durant une période donnée, et donc à ajouter de la précision.

Définition 4.6 (Découpage temporellement correct en-ligne)**.**
Soit OBS un ensemble d'automates. Soit $\mathcal{W} = (\mathcal{W}_1, \ldots, \mathcal{W}_i, \mathcal{W}_{i+1})$ *une séquence correcte de fenêtres temporelles. La chaîne d'automates* $(Obs^1, \ldots, Obs^i)$ *est un* découpage temporellement correct en-ligne *de OBS si* $\forall Obs \in OBS$, $\exists Obs'$ *tel que* $(Obs^1, \ldots, Obs^i, Obs')$ *est un découpage temporellement correct de Obs par rapport à* $\mathcal{W}$*.*

Le découpage temporellement correct en-ligne consiste à découper en-ligne l'automate pour produire une chaîne d'automates temporellement correcte. Notons $\forall j$, $\mathcal{W}_j = [t_{j-1}, t_j]$ ($\mathcal{W}_{i+1} = [t_i, t_n]$). Remarquons que nous n'avons pas nécessairement $t_i = t$, où t est la date à laquelle on effectue le découpage en-ligne. Si on a $t_i = t$, alors on recherche les comportements qui ont eu lieu jusqu'à la date actuelle. Si $t_i < t$, alors on effectue le diagnostic du passé. Enfin, si $t_i > t$, alors on effectue une prédiction de l'évolution du système. Le problème d'effectuer un diagnostic jusqu'à une date proche voire de faire une prédiction, c'est qu'il est possible de considérer que le système produise un nombre non borné d'observations.

4.3 Découpage des observations en-ligne

Nous montrons comment découper l'automate des observations pour les hypothèses que nous avons posées sur les canaux de communication. Nous considérons par la suite trois cas : le cas des fenêtres sûres, le cas des fenêtres non sûres, et le cas où l'on considère que des observations peuvent être en tampon.

4.3.1 Découpage par fenêtres sûres

Le concept de fenêtres sûres a été défini dans [PCR01]. Une date t (appelée *point d'arrêt*) est qualifiée de saine si toute observation émise avant t a été reçue avant t. Il est possible de repérer ces dates dans certains cas. En effet, on a généralement la propriété qu'une observation met un temps dt borné pour atteindre le superviseur. Ainsi, si aucune observation n'est reçue pendant une période suivant t d'une durée supérieure à dt, on est certain que l'ensemble des observations émises avant t est l'ensemble des observations reçues avant t.

Les fenêtres sont qualifiées de sûres si elles sont délimitées par des points d'arrêts sains.

Soit O l'ensemble des observations reçues à la date t et $t_1, \ldots, t_i$ une séquence de point d'arrêts sains telle que $t_i < t$. On considère que toute observation a été émise avant t_i. On note $\mathcal{W}_j = [t_{j-1}, t_j]$. Puisque les fenêtres temporelles sont sûres, il est possible de partitionner les observations en ensembles associés chacun à la fenêtre temporelle

pendant laquelle les observations ont été envoyées. On note O_j l'ensemble d'observations associées à la fenêtre temporelle $\mathcal{W}_j$. On note $O^j = O_1 \cup \cdots \cup O_j$.

Résultat 4.3 (Découpage en-ligne)**.**
Soit OBS l'ensemble des automates qui peut être construit à partir de O. Le découpage temporellement correct en-ligne de OBS est la chaîne d'automates $(Obs^1, \ldots, Obs^i)$ *définie comme suit.*

$\forall j$, $Obs^j = (Q^j, E^j, T^j, I^j, F^j)$ *avec :*
- $Q^j = \{q \mid O^{j-1} \subseteq q \subseteq O^j \wedge \lhd (q)\}$,
- $E^j = E_{Obs}$,
- T^j *est l'ensemble des transitions entre les états de* Q^j *défini comme dans la construction de l'automate des observations (résultat 2.3),*
- $I^j = \{O^{j-1}\}$, *et*
- $F^j = \{O^j\}$.

Exemple : Considérons que nous avons trois observations $A = (a, 1)$, $B = (b, 2)$ et $C = (c, 3)$ et deux points d'arrêts sains t_1 et t_2 tels que $A \prec' t_1$, $t_1 \prec' B \prec' t_2$ et $t_1 \prec' C \prec' t_2$. On a $O^0 = \emptyset$, $O_1 = \{A\}$, $O^1 = \{A\}$, $O_2 = \{B, C\}$ et $O^2 = \{A, B, C\}$. La chaîne d'automates de la figure 4.2 représente le découpage temporellement correct en-ligne.

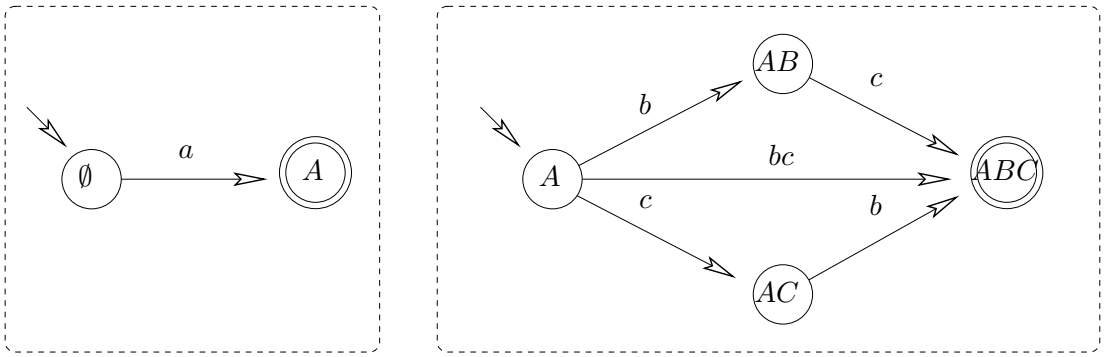

FIG. 4.2 – Découpage en-ligne pour les fenêtres sûres

4.3.2 Découpage sans fenêtre sûre

La difficulté de l'hypothèse des fenêtres sûres est double. D'une part, il faut être capable de déterminer ces fenêtres sûres. Dans le cas de systèmes de taille importante, il est réaliste de considérer qu'un flux constant d'observations parvient au superviseur, et qu'il n'est pas possible de trouver de points d'arrêt sains. Le second problème est que le superviseur ne choisit pas ses points d'arrêt, ce qui est particulièrement problématique dans le cas du diagnostic de systèmes reconfigurables (voir le chapitre 6). Nous proposons donc d'effectuer un découpage sans l'hypothèse de fenêtre sûre.

Soit O l'ensemble des observations reçues à la date t. Soit $t_1, \ldots, t_i$ une séquence de dates telle que toute observation émise avant t_i ait été reçue ($t_i < t$). On note $\mathcal{W}_j = [t_{j-1}, t_j]$. On note O_j l'ensemble des observations qui ont pu être émises pendant la période $\mathcal{W}_j$. On note $O^j = O_1 \cup \cdots \cup O_j$. L'ensemble O^j est l'ensemble des observations qui ont pu être émises avant t_j. De la même manière $P^j = O \setminus (O_{j+1} \cup \cdots \cup O_i)$ représente l'ensemble des observations certainement émises avant t_j. Remarquons que dans le cas des fenêtres sûres, on a $O^j = P^j$.

Résultat 4.4 (Découpage en-ligne)**.**
Soit OBS l'ensemble des automates qui peuvent être construits à partir de O. Le découpage temporellement correct en-ligne de OBS est la chaîne d'automates $(Obs^1, \ldots, Obs^i)$ *définie comme suit.*

$\forall j,\ Obs^j = (Q^j, E^j, T^j, I^j, F^j)$ *avec :*
- $Q^j = \{q \mid P^{j-1} \subseteq q \subseteq O^j \wedge \lhd (q)\}$,
- $E^j = E_{Obs}$,
- T^j *est l'ensemble des transitions entre les états de* Q^j *défini comme dans la construction de l'automate des observations (résultat 2.3),*
- $I^j = \{q \mid P^{j-1} \subseteq q \subseteq O^{j-1} \wedge \lhd (q)\}$, *et*
- $F^j = \{q \mid P^j \subseteq q \subseteq O^j \wedge \lhd (q)\}$.

Exemple : Reprenons notre exemple précédent en considérant que $t_1 \not\prec' B$. Cela signifie qu'on ne sait pas si l'observation B a été émise après t_1, mais cela ne signifie pas forcément que l'observation a été émise avant t_1. Alors, l'ensemble $O^1 = \{A, B\}$ (resp. $O^2 = \{A, B, C\}$) correspond à l'ensemble maximum d'observations émises à la date t_1 (resp. t_2). L'ensemble $P_1 = \{A\}$ (resp. $P_2 = \{A, B, C\}$) est l'ensemble minimum d'observations émises à la date t_1 (resp. t_2). La figure 4.3 donne l'automate des observations pour cet exemple.

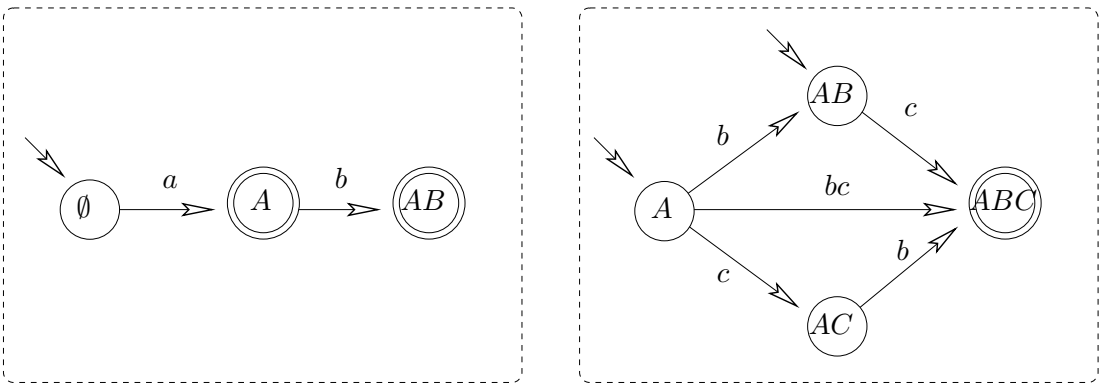

FIG. 4.3 – Découpage en-ligne pour les fenêtres non sûres

4.3.3 Découpage avec hypothèses sur les observations en tampon

Lors du précédent point, nous avons considéré que toutes les observations émises avant t_i avaient été reçues avant t. Cela signifie que le découpage en-ligne permet d'obtenir une chaîne d'automates qui décrit les observations émises jusqu'à la date t_i. Ainsi, le diagnostic peut expliquer le comportement du système jusqu'à la date t_i. Une extension serait de supprimer l'hypothèse sur t_i pour obtenir, par exemple, $t_i = t$ (diagnostic du système jusqu'à la date t) ou $t_i > t$ (prédiction de l'évolution future du système).

Le problème est alors le suivant : la séquence d'observations émises pendant la période $[t_{i-1}, t_i]$ était jusqu'alors bornée par l'ensemble des observations reçues ; ce n'est plus le cas si $t_i > t - dt$. Ainsi, le système peut avoir émis une séquence d'observations de taille non bornée pendant la période $[t_{i-1}, t_i]$ sans que ces observations aient été reçues. Alors, le nombre d'états du morceau d'automate Obs^i n'est pas borné.

Il est donc nécessaire d'effectuer l'hypothèse suivante :

Hypothèse 4.1.
Le nombre maximal d'observations émises pendant la période $[t_0, t_i]$ *est* $\max_i$.

Nous nous plaçons maintenant dans un cas particulier. Nous considérons que $t_{i-1} < t - dt$. La raison de cette hypothèse est qu'il n'est pas intéressant de raisonner sur une période $[t_{i-1}, t_i]$ si on ne sait déjà pas quelles observations ont été émises avant t_{i-1}. Nous nous plaçons également dans le cas où $t_i \geq t$. Grâce à cette hypothèse, on peut considérer que toutes les observations reçues à t ont été émises à t_i. Dans le cas contraire, il ne faudrait pas considérer que l'ensemble q des observations reçues à la date t est l'ensemble minimum mais un sous-ensemble $q' \subseteq q$.

Soit q l'ensemble des observations reçues à la date t. Remarquons que le nombre d'observations émises non encore reçues à la date t est au maximum $n_i = \max_i - \mid q \mid$.

L'ensemble des observations émises à la date t_i est un sur-ensemble de q puisque $t_i \geq t$. q est le plus petit ensemble des observations ayant pu être émis à la date t. Donc, $q \in F^i$ l'ensemble des états finaux de Obs^i.

La notation des états était jusqu'alors un ensemble d'observations (par le résultat 2.3). Cependant, cette notation est plus difficile à utiliser. D'une part, nous ne connaissons pas à l'avance les *id* des observations. Mais ce point peut être contourné en considérant que les *id* ne sont pas encore instanciés. D'autre part, un problème est celui montré à la figure 4.4. Dans cet exemple, on n'a reçu aucune observation mais on considère que deux observations peuvent avoir été émises. Le premier automate présente (en partie) cette possibilité en considérant que les observations A et B ont pu être émises. Par la suite, on apprend qu'effectivement A et B ont été émises et de plus qu'elles ont été émises avant t. On sait que $A \prec B$. Alors, on obtient le second automate de la chaîne. Le problème qui apparait ici est que cette chaîne inclut la possibilité que B ait été émis avant A.

Aussi, nous construisons le morceau d'automate Obs^i en utilisant une technique à la frontière des deux techniques proposées dans le résultat 2.2 et dans le résultat 2.3.

Nous représentons les états par un ensemble e d'observations et une séquence seq d'événements observables. L'ensemble e est tel que toute observation de e précède toute

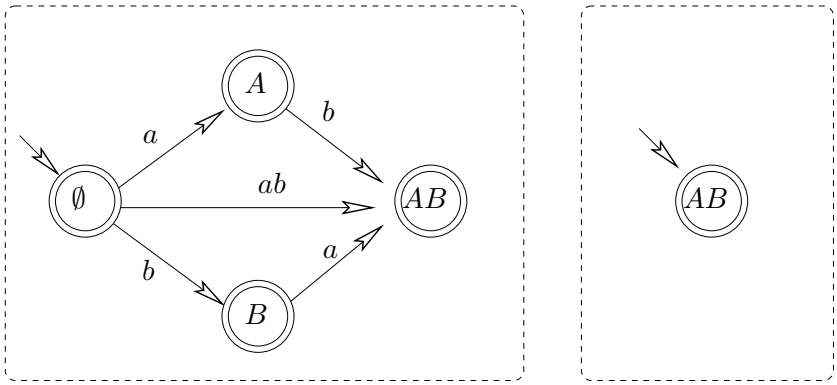

FIG. 4.4 – Découpage en-ligne avec hypothèses sur les observations manquantes

observation de seq dans les trajectoires menant à cet état, et e ne contient aucune observation qui n'a pas été reçue à la date t_i. Par exemple, l'état $q = \{o, o'\}, (\{a\}, \{a, b\})$ indique que o et o' ont été reçus et que a puis a et b simultanément ont été émis par la suite. Puisque a ne fait pas partie de l'ensemble des observations, cela signifie que a n'a pas encore été reçu. Cet état fait l'hypothèse que a puis a et b ont été émis. Notons que l'événement observable b peut avoir été reçu mais il n'est pas ajouté à l'ensemble car b a été émis après le premier a qui est dans la séquence.

Les transitions sont définies comme dans les résultats 2.2 et 2.3. Remarquons qu'une fois que la séquence seq est non vide, aucune transition ne peut mener à un état contenant un ensemble d'observations plus grand.

Puisqu'on a nommé un certain nombre d'états finaux de manière particulière, il est nécessaire de tenir compte du nom donné à ces états dans la fenêtre suivante. Ainsi, certains états de Obs^{i+1} doivent être renommés pour qu'il portent le même nom que dans Obs^i.

Exemple : Considérons un système pouvant émettre les événements observables a, b, c et d. Pour cet exemple, nous ne considérons pas le cas de l'émission simultanée de plusieurs observations pour simplifier la figure. Considérons que $B = (b, 1)$ a été reçu mais qu'une observation supplémentaire a pu être émise. Le découpage est présenté sur le premier automate de la figure 4.5. Dans cette figure, les états sont notés de manière simplifiée. Ainsi, l'état Ba représente en réalité l'état $\{B\}, (\{a\})$, et l'état ab représente $\{\}, (\{a\}, \{b\})$. Sur cet automate, on voit qu'une transition étiquetée par b part du premier état qui correspond à la possibilité que B ait été émis avant toute autre observation. Puisque B a été effectivement reçu, on ajoute B à l'ensemble des observations. À partir de l'état $\{B\}, ()$, on peut tirer la transition étiquetée par a. Puisque cette observation n'a pas été reçue, on ajoute $\{a\}$ à la séquence pour obtenir l'état cible.

Retournons à l'état initial et tirons à présent la transition étiquetée par c. Cette observation n'a pas été reçue, et on atteint donc l'état $\{\}, (\{c\})$. Cet état n'est pas final

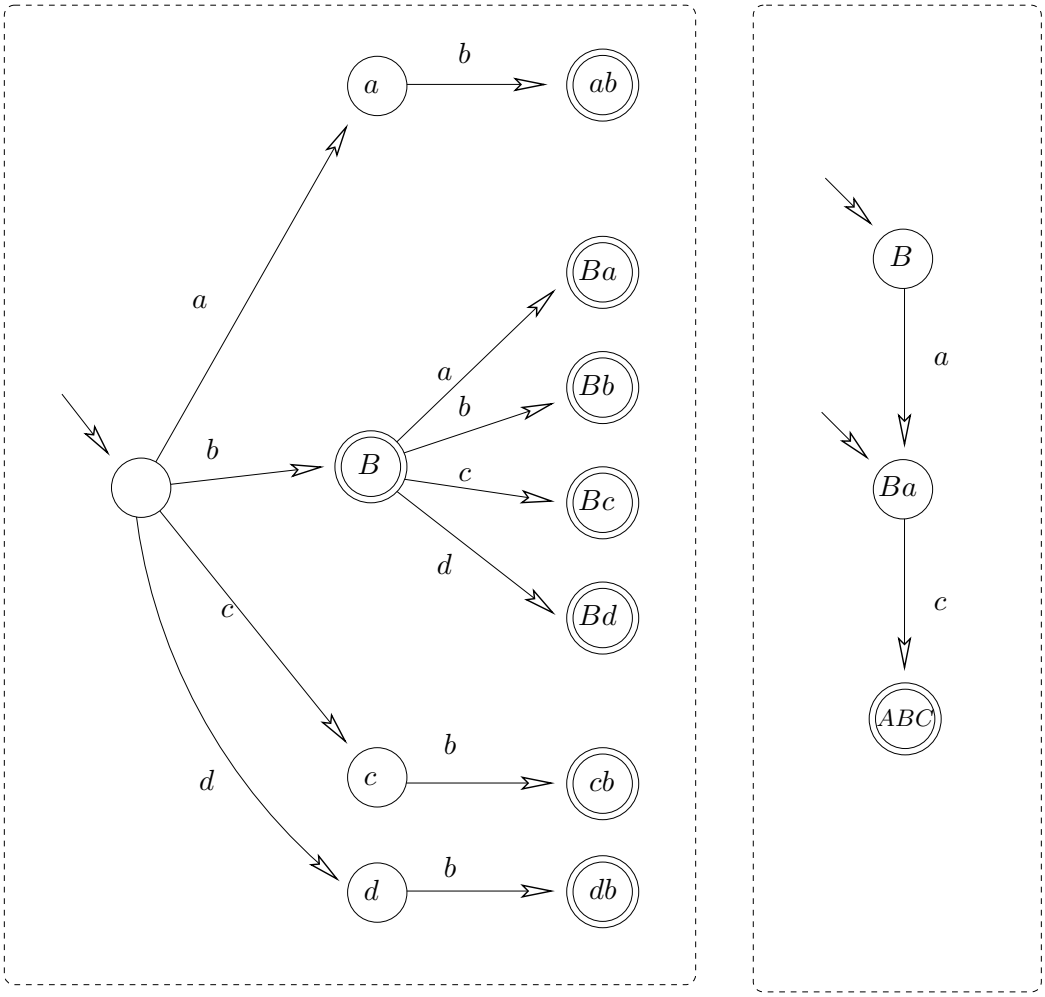

FIG. 4.5 – Automate avec hypothèses sur les observations non reçues

puisque l'observation B n'y est pas expliquée. La seule transition partant de cet état est étiquetée par b (puisqu'il ne peut y avoir qu'une seule observation non reçue). Puisque la séquence n'est pas vide, on n'atteint pas $\{B\}, (\{c\})$ mais $\{\}, (\{c\}, \{b\})$ (l'observation est ajoutée à la séquence).

Dans la fenêtre suivante, on reçoit l'observation $A = (a, 2)$ avec $B \prec A$ mais $t_i \not\prec' A$. Ainsi, soit B, soit B puis A ont été émis durant la première fenêtre temporelle. Il y donc deux états initiaux : $\{B\}, ()$ et $\{B\}, (\{a\})$. En outre, le superviseur reçoit l'observation $C = (c, 3)$. On considère qu'aucune autre observation n'a pu être émise. Alors, les observations A, B et C ont toutes été reçues avant le diagnostic, et donc l'état final est $q = \{A, B, C\}$ (et non pas $\{B\}, (\{a\}, \{c\})$.

Remarquons que cette méthode produit un nombre d'états exponentiel par rapport au nombre n_i d'observations non reçues. Aussi, cette méthode est difficile à utiliser si ce nombre est grand.

4.4 Conclusion

Dans ce chapitre, nous avons considéré le diagnostic en-ligne de systèmes à événements discrets. Effectuer un diagnostic en-ligne consiste à calculer le diagnostic pendant que le système émet les observations. Ainsi, il faut résoudre deux difficultés : comment calculer le diagnostic de manière incrémentale pour éviter un temps de calcul trop important et comment raisonner à partir d'observations incomplètes ?

Nous avons résolu la première difficulté au chapitre 3 grâce au diagnostic par chaînes d'automates. Pour cela, nous considérons l'ensemble des automates des observations possibles à l'issue du diagnostic étant données les observations actuelles. Nous avons défini un *découpage en-ligne* qui consiste à calculer une chaîne d'automates qui soit partielle pour chaque automate des observations. Ainsi, quelque soit l'automate des observations que nous construirons par la suite, le découpage de cet automate est déjà partiellement connu. Aussi, il est possible de raisonner sur cette chaîne partielle et de réutiliser (grâce aux résultats du chapitre 3) le raisonnement déjà effectué lors du diagnostic final.

D'autre part, le découpage en-ligne effectué à une date ultérieure est la même chaîne à laquelle on a ajouté un automate. Ainsi, lors du raisonnement sur cette seconde chaîne, il est possible de réutiliser (encore une fois grâce aux résultats du chapitre 3) le raisonnement effectué sur la première chaîne.

La seconde difficulté est résolue en prenant en considération toutes les observations qui ont pu être émises et non reçues. Nous avons montré comment résoudre ce problème dans notre contexte, mais dans le cas d'hypothèses différentes sur les observations, les méthodes proposées doivent être adaptées ou changées.

Chapitre 5

Diagnostic décentralisé par chaînes d'automates

Ce chapitre considère deux notions qui sont très proches : le calcul décentralisé et le diagnostic décentralisé, et ce dans le contexte de diagnostic par chaîne d'automates que nous avons introduit au cours du chapitre 3. Ces deux notions s'appuient généralement (bien que pas nécessairement, voir par exemple [WYL04]) sur une modélisation décentralisée.

Le calcul décentralisé du diagnostic est une technique consistant à effectuer des calculs de diagnostic locaux à des sous-systèmes avant de *fusionner* ces diagnostics pour obtenir le diagnostic global du système. Cette technique permet de restreindre l'espace de recherche du diagnostic global. Le diagnostic décentralisé consiste à conserver certains diagnostics locaux pour éviter l'explosion combinatoire lors de la fusion des diagnostics locaux.

Ce chapitre est divisé comme suit. Tout d'abord, nous montrons comment calculer de manière décentralisée le diagnostic global ou décentralisé du système par automates. Enfin, nous voyons le calcul décentralisé du diagnostic, puis le calcul du diagnostic décentralisé dans le contexte de diagnostic par chaîne d'automates.

5.1 Calcul et diagnostic décentralisés par automates

Dans cette section, nous expliquons le calcul décentralisé et le diagnostic décentralisé pour les définitions de diagnostic que nous avons données dans le chapitre 2. Pour cela, nous présentons d'abord la modélisation décentralisée. Ensuite, nous montrons comment appliquer le calcul décentralisé du diagnostic et présentons le diagnostic décentralisé.

5.1.1 Modélisation décentralisée

Les approches décentralisées sont généralement utilisées dans le contexte de systèmes décentralisés *par nature*, c'est-à-dire de systèmes constitués de composants ou de fonctions interagissant. Le comportement du système est alors complètement défini par

le modèle de chacun de ses composants, et la description des liens entre les composants si ceux-ci ne sont pas implicites dans les modèles des composants.

Définition 5.1 (Modèle décentralisé)**.**
Le modèle décentralisé *d'un système est un ensemble de modèles locaux* $\{Mod_1, \ldots, Mod_m\}$ *où* $\forall i$, Mod_i *est le modèle du composant* c_i.

Le modèle de chaque composant est un automate $Mod_i = (Q_i, E_i, T_i, I_i, F_i)$. Dans ce chapitre, l'indice i fait référence au composant i, contrairement aux chapitres précédents. Il est possible d'appliquer les techniques que nous présentons pour des systèmes non décentralisés, mais il est dans ce cas nécessaire de définir des *composants abstraits*.

Ce modèle du système est qualifié d'implicite. Le modèle explicite est défini comme suit :

Définition 5.2 (Modèle explicite)**.**
Le modèle explicite *du modèle décentralisé* $\{Mod_1, \ldots, Mod_m\}$ *est défini par :* $Mod = Mod_1 \otimes \cdots \otimes Mod_m$.

Le modèle explicite est de taille exponentielle par rapport au nombre m de composants du système. Aussi, nous manipulons généralement le modèle décentralisé.

5.1.2 Calcul décentralisé par automate

Le calcul décentralisé du diagnostic consiste à effectuer un diagnostic de manière locale à chaque composant, puis à fusionner les diagnostics entre eux de manière à obtenir le diagnostic global du système. Le principal intérêt de cette approche est qu'il n'est pas nécessaire de calculer le modèle global du système, qui n'est pas calculable pour des systèmes qui nous intéressent.

Nous explicitons ici le calcul décentralisé dans le cas où l'automate des observations peut se définir comme une synchronisation d'automates locaux, puis nous donnons un bref aperçu d'une généralisation possible.

5.1.2.1 Cas simple

Nous considérons dans ce cas que l'automate des observations Obs peut se définir comme une synchronisation de m automates Obs_i : $Obs = Obs_1 \otimes \cdots \otimes Obs_m$. Généralement, on considère que Obs_i contient les observations du composant c_i et d'aucun autre composant, quand c'est possible. Dans le cas contraire, on risque de ne pas avoir un résultat performant.

Grâce aux résultats donnés à la section A.4, nous aboutissons à ce résultat :

$$\begin{aligned} \Delta &= Mod \otimes Obs \\ &= (Mod_1 \otimes \cdots \otimes Mod_m) \otimes (Obs_1 \otimes \cdots \otimes Obs_m) \\ &= (Mod_1 \otimes Obs_1) \otimes \cdots \otimes (Mod_m \otimes Obs_m) \end{aligned}$$

On note $\Delta_i = Mod_i \otimes Obs_i$. Δ_i est appelé le diagnostic local au composant c_i. Il s'agit du diagnostic du composant lorsqu'on ne considère pas ses interactions possibles avec les autres composants.

Le calcul du diagnostic global est présenté comme la synchronisation de tous les diagnostics locaux, mais il est possible d'effectuer cette synchronisation deux à deux. Par exemple, $\Delta = (\Delta_1 \otimes \Delta_2) \otimes \cdots \otimes \Delta_m$. On dit que $\Delta_1 \otimes \Delta_2$ est la *fusion* des deux diagnostics (voir [PC05]).

Le but du calcul décentralisé est de restreindre fortement (à l'aide des observations) les comportements locaux (grâce au calcul de Δ_i) avant de fusionner les comportements. De cette manière, on évite l'explosion combinatoire qu'on obtient lors du calcul du modèle explicite.

5.1.2.2 Généralisation

Considérons un système constitué de composants. Considérons qu'il y a un capteur par composant, que ces capteurs ne sont pas synchronisés et qu'on ne connait pas les délais d'émission d'une observation vers le superviseur. Alors, il n'est pas possible de donner l'ordre d'émission entre une observation provenant d'un composant et une observation provenant d'un autre composant. Aussi, la propriété $\exists Obs_1, \ldots, Obs_m$ tel que $Obs = Obs_1 \otimes \cdots \otimes Obs_m$ est évidente.

Si on considère qu'on dispose d'une borne supérieure du délai de transmission d'une observation, alors on peut ordonner certaines observations provenant de capteurs différents. La propriété considérée jusque là n'est pas forcément vraie. On peut considérer différentes manières de conserver une approche décentralisée dans ce contexte.

La première méthode est d'effectuer le diagnostic avec $Obs' = Obs_1 \otimes \cdots \otimes Obs_m \neq Obs$. Dans ce cas, nous considérons ici plus de comportements que ce qu'il faudrait considérer. Cependant, il est réaliste de penser que les comportements supplémentaires ne sont que des comportements équivalents à un ordonnancement des événements près. Si cela est vrai, le *diagnostic*, défini comme l'ensemble des événements de panne ayant eu lieu sur le système, est identique. Notons que dans les approches distribuées, on ne prend souvent pas en compte le fait que les observations provenant de différents capteurs sont ordonnables (voir par exemple [SW04]).

Une deuxième méthode est de considérer que les automates des observations locaux disposent d'événements leur permettant de se synchroniser entre eux. Ainsi, l'exemple de la figure 5.1 montre deux automates représentant l'émission des observations a et b sur deux composants différents. La synchronisation des deux automates indique que a a eu lieu avant b. Ici, z n'est pas un événement observable. Cette méthode semble cependant très difficile à mettre en place pour un nombre important d'automates.

On peut également considérer qu'on a la propriété suivante : $Obs = Obs_1 \otimes \cdots \otimes Obs_m \otimes Obs_{ss_1} \otimes \cdots \otimes Obs_{ss_k}$ où Obs_{ss_i} représente les observations émises par le sous-système ss_i. Ainsi, le diagnostic local Δ_i est toujours défini par $Mod_i \otimes Obs_i$, mais le diagnostic du sous-système ss_i constitué des composants j_1 à j_p est alors défini ainsi : $\Delta_{j_1} \otimes \cdots \otimes \Delta_{j_p} \otimes Obs_{ss_i} \otimes Obs_{sss_1} \otimes \cdots \otimes Obs_{sss_k}$ où $Obs_{sss_1} \otimes Obs_{sss_k}$ représente les automates des observations des sous-systèmes de ss_i.

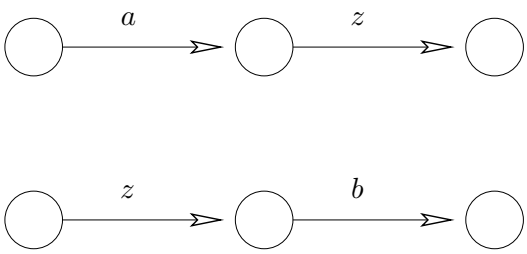

FIG. 5.1 – Synchronisation d'observations sur deux automates locaux

Enfin, on peut considérer que Obs est un *sous-automate* de $Obs_1 \otimes \cdots \otimes Obs_m$, c'est-à-dire l'automate $Obs_1 \otimes \cdots \otimes Obs_m$ auquel on a supprimé certaines transitions correspondant à des ordres d'émission impossibles.

Considérons l'exemple de deux composants et soit Obs un sous-automate de $Obs_1 \otimes Obs_2$. On note $\Delta_1 = Mod_1 \otimes Obs_1$ et $\Delta_2 = Mod_2 \otimes Obs_2$. De plus, on note $\Delta = (Mod_1 \otimes Mod_2) \otimes Obs$. Alors, Δ est un sous-automate de $\Delta_1 \otimes \Delta_2$. De plus, Δ est l'automate $\Delta_1 \otimes \Delta_2$ dont on a supprimé toutes les transitions $((q_{Obs_1}, q_{Obs_2}, q_{Mod_1}, q_{Mod_2}), l_{Obs} \cup l_{Mod}, (q'_{Obs_1}, q'_{Obs_2}, q'_{Mod_1}, q'_{Mod_2}))$ telles que $(((q_{Obs_1}, q_{Obs_2}), l_{Obs}, (q'_{Obs_1}, q'_{Obs_2}))$ apparait dans $Obs_1 \otimes Obs_2$ mais pas dans Obs. Il s'agit donc de retirer certaines transitions au moment de la fusion des diagnostics locaux.

Des quatre propositions que nous venons d'énumérer, la dernière nous semble la plus intéressante parce qu'elle ne perd pas d'information contrairement à la première proposition, elle ne nécessite pas une construction complexe contrairement à la deuxième proposition et qu'elle ne nécessite pas beaucoup plus de calcul contrairement à la troisième proposition.

Cependant, par la suite, nous laissons ce point de côté et considérons pour simplifier la propriété $Obs = Obs_1 \otimes \cdots \otimes Obs_m$.

5.1.3 Diagnostic décentralisé par automate

Le diagnostic décentralisé a été défini dans [PC05]. Nous présentons tout d'abord succinctement la façon dont nous le définissons. Nous montrons ensuite un algorithme permettant de calculer le diagnostic décentralisé. Nous montrons alors comment il est possible de mettre en évidence les comportements indépendants des sous-systèmes. Enfin, la dernière partie est consacrée à la principale limite de cette méthode.

5.1.3.1 Principe

Le but du diagnostic décentralisé est de considérer que certains sous-systèmes agissent de manière indépendante, c'est-à-dire qu'ils n'interagissent pas les uns avec les autres. Dès lors, fusionner leurs diagnostics n'apporte aucune information supplémentaire. Cette décision est dynamique. On peut ainsi considérer chaque sous-système comme un système.

Définition 5.3 (Indépendance de comportement)**.**
Soit deux sous-systèmes ss_1 et ss_2 disjoints. Soit Δ_{ss_1} et Δ_{ss_2} les diagnostics des deux sous-systèmes. On dit que Δ_{ss_1} et Δss_2 sont indépendants *si les étiquettes de transition de Δ_{ss_1} et Δ_{ss_2} ne comportent pas d'événements de synchronisation des deux sous-systèmes.*

Par abus de langage, on dit que les sous-systèmes sont indépendants si leurs diagnostics sont indépendants. ss_1 et ss_2 sont indépendants si leurs comportements ne peuvent pas se synchroniser directement. Notons que cette définition est différente de la définition d'indépendance de diagnostic présentée dans [PC05]. Ici, dans la définition que nous avons donnée, deux sous-systèmes peuvent se synchroniser *via* un sous-système intermédiaire.

Définition 5.4 (Diagnostic décentralisé)**.**
Soit Γ un système modélisé par Mod et Obs l'automate des observations sur le système. Soit $\Delta = Mod \otimes Obs$ le diagnostic du système. Le diagnostic décentralisé *du système est une double séquence $((ss_1, \ldots, ss_k), (\Delta_{ss_1}, \ldots, \Delta_{ss_k}))$ telle que*
- *$(ss_1, \ldots, ss_k)$ est un partitionnement du système en sous-systèmes,*
- *$\forall i$, Δ_{ss_i} est le diagnostic du sous-système ss_i,*
- *les sous-systèmes ss_i sont indépendants deux à deux,*
- *$\Delta = \Delta_{ss_1} \otimes \cdots \otimes \Delta_{ss_k}$.*

Le diagnostic décentralisé $((ss_1, \ldots, ss_k), (\Delta_{ss_1}, \ldots, \Delta_{ss_k}))$ peut être noté $(\Delta_{ss_1}, \ldots, \Delta_{ss_k})$, les sous-systèmes étant sous-entendus. On oppose le diagnostic décentralisé au diagnostic dit *global*.

L'avantage d'un diagnostic décentralisé par rapport à un diagnostic global est très important lorsqu'on parvient à trouver des sous-systèmes indépendants. Considérons que le nombre d'états des diagnostics Δ_i est n_i. Alors, le nombre d'états du diagnostic global Δ vaut $n_1 \times \cdots \times n_k$. La taille du diagnostic décentralisé est donc très fortement inférieure à la taille du diagnostic global. Par ailleurs, la compréhension du diagnostic par un opérateur humain est simplifiée puisque le comportement du système est divisé en comportements plus simples.

5.1.3.2 Calcul du diagnostic décentralisé

Pour calculer un diagnostic décentralisé, il convient de remarquer quels sous-systèmes ont un comportement indépendant. Cette tâche n'est pas forcément évidente et nous proposons plusieurs méthodes.

Utiliser le diagnostic global. Une première méthode est de construire le diagnostic global Δ pour isoler les sous-systèmes indépendants. L'algorithme 2 présente comment il est possible de construire un sous-système indépendant à partir du diagnostic d'un sous-système. Étant donné un composant c, cet algorithme calcule l'ensemble des composants qui interagissent avec ce composant ou indirectement par un ou plusieurs autres composants.

Algorithme 2 Isoler un sous-système indépendant

Entrée : Diagnostic Δ du (sous-)système ss, composant c
Sous-système $ss' := \{c\}$
Tant que il existe une transition de Δ étiquetée par un événement e du sous-système ss' et de $(ss - ss')$ **faire**
  $ss' = ss' \cup$ l'ensemble des composants qui se synchronisent sur e
Fin tant que
rendre ss'

Une fois qu'on connait un sous-système ss' indépendant, il est possible de projeter le diagnostic Δ sur ss'. Remarquons que $(ss - ss')$ est également un sous-système indépendant et il est possible de construire le diagnostic de ce sous-système également. On obtient alors le diagnostic décentralisé $(\Delta_{ss'}, \Delta_{ss-ss'})$. Il est alors possible de recommencer cette opération sur le sous-système $ss - ss'$.

Remarquons que cette méthode permet de construire le diagnostic décentralisé le meilleur qui soit (c'est-à-dire avec une partition maximale du système). Cependant, elle ne permet d'atteindre que partiellement les objectifs du diagnostic décentralisé. En effet, pour obtenir le diagnostic décentralisé, il est nécessaire d'avoir au préalable calculé le diagnostic global, ce que nous souhaitons éviter.

Calcul et diagnostic décentralisés Il paraît logique de calculer le diagnostic décentralisé de manière décentralisée. Le principe est d'interdire la fusion des diagnostics de sous-systèmes indépendants. Remarquons cependant que si deux sous-systèmes ss_1 et ss_2 sont indépendants, ils peuvent être tous les deux dépendants d'un troisième sous-système ss_3. Dans ce cas, le diagnostic de ss_1 est fusionné avec le diagnostic de ss_3, et le résultat est ensuite fusionné avec ss_2. Lorsqu'aucune paire de sous-systèmes dépendants ne peut plus être calculée, alors le diagnostic décentralisé est l'ensemble des diagnostics locaux aux sous-systèmes obtenus.

Le problème qui se pose alors est de découvrir avant la fusion que les sous-systèmes sont indépendants. Il est également possible d'effectuer la fusion puis de vérifier l'indépendance, mais il est plus efficace d'éviter la fusion si possible. Le point 5.1.3.3 s'intéresse au problème de trouver des sous-systèmes indépendants.

L'algorithme de fusion de diagnostic reprend alors l'algorithme classique avec la différence suivante. Après la fusion de deux diagnostics, il faut vérifier s'il existe deux sous-systèmes indépendants au sein du sous-système grâce à l'algorithme 2. Si on obtient un ou plusieurs sous-systèmes, il convient de projeter le diagnostic global du sous-système pour obtenir plusieurs diagnostics locaux indépendants.

5.1.3.3 Trouver des sous-systèmes indépendants

Il n'est parfois pas évident de trouver des comportements indépendants. Ainsi, considérons la figure 5.2. Dans cet exemple, les trois automates en haut de la figure représentent les diagnostics locaux de trois composants. Les étiquettes a_1 à a_3 représentent

des événements exogènes tandis que les événements c_{ij} correspondent à des événements de synchronisation entre le composant i et le composant j. Si on considère les diagnostics deux à deux, on pense qu'il y a une dépendance de comportement. Cependant, si on fusionne les deux premiers diagnostics (automate au centre de la figure) et si on le fusionne avec le troisième diagnostic, on voit que les composants n'ont pas communiqué pendant la période considérée. Ainsi, dans l'exemple, les diagnostics obtenus par abstraction du diagnostic global ne comportent chacun qu'un seul état (A, B ou C).

Il est intéressant d'être capable de déterminer que deux sous-systèmes sont indépendants *avant* de fusionner leurs diagnostics. Ceci permet d'éviter la fusion de deux diagnostics indépendants, et de réduire la taille des diagnostics produits. Cependant, il n'est pas forcément évident de déterminer cette propriété de manière immédiate, et il est parfois nécessaire d'effectuer la fusion (ou un raisonnement comparable à la fusion) pour déterminer que deux sous-systèmes sont indépendants.

Nous proposons cependant quelques points pour déterminer que deux sous-systèmes sont indépendants. Deux sous-systèmes sont indépendants s'ils n'ont pas échangé de messages, c'est-à-dire s'il n'existe aucun événement permettant de synchroniser les diagnostics Δ_{ss_1} et Δ_{ss_2} des deux sous-systèmes ss_1 et ss_2. Cela peut se traduire de diverses manières :

- Les ensembles d'événements des automates de diagnostic E_{ss_1} et E_{ss_2} sont disjoints. Dans ce cas, les composants ne communiquent jamais entre eux. Ils sont évidemment indépendants.
- Les ensembles d'événements des automates de diagnostic E_{ss_1} et E_{ss_2} ont une intersection E non vide, mais aucun de ces événements n'étiquette une transition de Δ_{ss_1} ni Δ_{ss_2}. On sait qu'aucun message n'a transité entre les deux sous-systèmes.
- Les ensembles d'événements des automates de diagnostic E_{ss_1} et E_{ss_2} ont une intersection E non vide et certains de ces événements étiquettent des transitions de Δ_{ss_1} et de Δ_{ss_2}, mais les événements étiquetant les transitions des diagnostics Δ_{ss_1} et Δ_{ss_2} sont disjoints. Cela signifie qu'au moins un des diagnostics émet une hypothèse de communication *via* un événement de synchronisation c, mais que cette hypothèse n'est pas confirmée par le second diagnostic. Il est alors possible, de supprimer toute transition de Δ_{ss_1} et Δ_{ss_2} qui contient un événement de synchronisation de E. Remarquons qu'à l'issue de cette suppression, on peut vérifier si chaque sous-système ne peut être à nouveau subdivisé en sous-systèmes indépendants. Par ailleurs, il est théoriquement nécessaire de mettre à jour les informations concernant les diagnostics ainsi modifiés puisqu'on peut découvrir à l'issue de cette modification que le sous-système est indépendant d'un autre sous-système.

Nous n'avons pas cherché d'autres méthodes permettant de trouver des comportements indépendants, mais il s'agit d'un point de recherche intéressant.

5.1.3.4 Limites de cette méthode

L'utilisation du diagnostic décentralisé n'est pas toujours suffisante pour empêcher l'explosion du nombre d'états lors du calcul du diagnostic global. En particulier, on

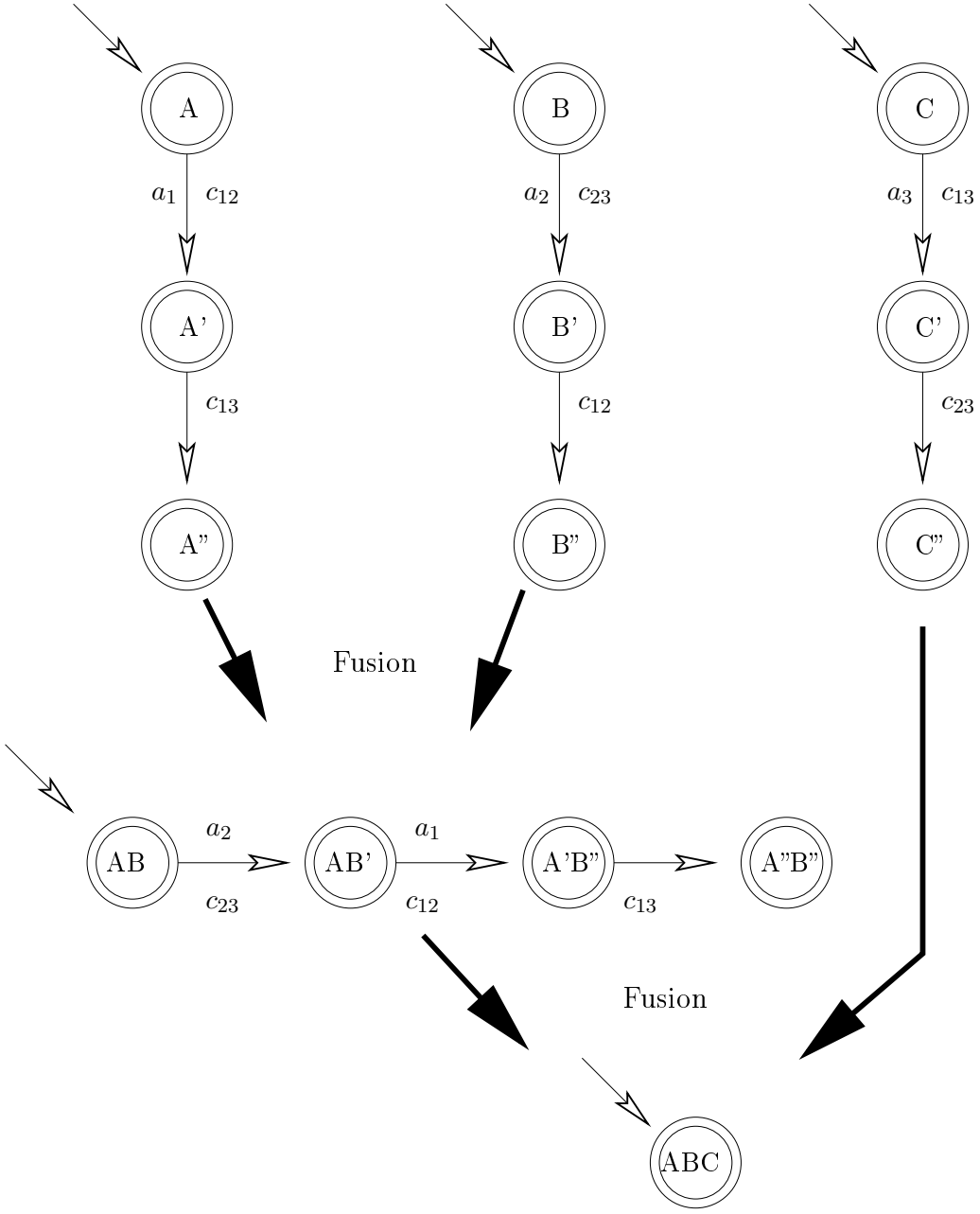

FIG. 5.2 – Exemple de sous-systèmes dépendants

remarque que plus la période à diagnostiquer est longue, plus il est difficile de diviser le système en sous-systèmes indépendants. En conséquence, sur une période longue, le diagnostic décentralisé revient à un diagnostic global. Ce point est également problématique dans le cadre du diagnostic en-ligne, puisque la durée du diagnostic augmente régulièrement, obligeant à fusionner les diagnostics de sous-systèmes de plus en plus gros. Aussi, nous proposons d'utiliser les chaînes d'automates pour découper la période de diagnostic en petites périodes permettant de mettre en lumière des comportements indépendants.

5.2 Calcul décentralisé du diagnostic par chaînes d'automates

5.2.1 Diagnostic par morceaux décentralisé

Le calcul décentralisé du diagnostic par morceaux est extrêmement simple. Soit $\{Mod_1, \ldots, Mod_m\}$ le modèle décentralisé du système et soit Obs l'automate des observations. Soit $(Obs^1, \ldots, Obs^n)$ un découpage correct de Obs. Soit $\forall i \in \{1, \ldots, n\}$, $\forall j$, Obs^i_j l'automate des observations du composant c_j pendant la fenêtre $\mathcal{W}_i$: $Obs^i = Obs^i_1 \otimes \cdots \otimes Obs^i_m$. Alors, le diagnostic par morceaux du système peut se calculer de la manière suivante :

Résultat 5.1. *$\Delta = Sli^{-1}(\mathcal{E}_\Delta)$ avec $\mathcal{E}_\Delta = (\Delta^1, \ldots, \Delta^n)$ où $\forall i \in \{1, \ldots, n\}$, $\Delta^i = (Mod^-_1 \otimes Obs^i_1) \otimes \cdots \otimes (Mod^-_m \otimes Obs^i_m)$.*

On note $\Delta^i_j = Mod^-_j \otimes Obs^i_j$ le diagnostic du composant c_j durant la fenêtre $\mathcal{W}_i$. On a alors : $\Delta^i = \Delta^i_1 \otimes \cdots \otimes \Delta^i_m$.

5.2.2 Diagnostic incrémental décentralisé

Nous avons vu que le diagnostic par morceaux avait l'inconvénient de générer un grand nombre d'états et de transitions inutiles. Aussi, nous adaptons la méthode au diagnostic incrémental.

Il est possible d'utiliser trivialement la formule de diagnostic incrémental suivante : $\mathcal{E}_\Delta = (\Delta^1, \ldots, \Delta^n)$ où $\forall i > 1$, $\Delta^i_\odot = (Mod^- \otimes Obs^i)[F^{i-1}_\Delta]$. Ainsi, on obtient : $\Delta^i_\odot = ((Mod^-_1 \otimes Obs^i_1) \otimes \cdots \otimes (Mod^-_m \otimes Obs^i_m))[F^{i-1}_\Delta]$. Ce résultat sous-entend cependant que la synchronisation des diagnostics locaux $\Delta^i_j = (Mod^-_j \otimes Obs^i_j)$ est effectuée *avant* la restriction (et non pas en même temps). De cette manière, on obtient un automate très grand que l'on restreint ensuite. Nous donnons une autre manière de procéder.

Pour cela, il est nécessaire de définir un opérateur permettant de transformer l'espace des états. La projection permet de considérer l'état d'un sous-système ou d'un composant.

Définition 5.5 (Projection).
Soit Q, un ensemble d'états q d'un système ($q = (q_1, \ldots, q_m)$ tel que q_i est un état de

l'automate A_i). On note $Q \downarrow \{A_{j1}, \ldots, A_{jp}\}$, appelé projection *de l'ensemble d'états Q sur $\{A_{j1}, \ldots, A_{jp}\}$, l'ensemble des états $(q_{j1}, \ldots, q_{jp})$ avec $\forall k$, q_{jk} est un état de A_{jk}, tel qu'il existe $q \in Q$ avec $q = (q_1, \ldots, q_{j1}, \ldots, q_{jp}, \ldots, q_m)$.*

Soit $q = (q_1, \ldots, q_n)$ l'état du système et soit le sous-système ss constitué des composants $c_{j1}, \ldots, c_{jp}$. Alors, la projection $q \downarrow \{Mod_{j1}, \ldots, Mod_{jp}\}$ est l'état du sous-système ss.

De même, soit Δ_1 et Δ_2 les diagnostics de deux sous-systèmes ss_1 et ss_2 tels que ss_1 est constitué des composants $c_{j1}, \ldots, c_{jp}$. Soit q_1 un état de Δ_1 et q_2 un état de Δ_2, et soit $q = (q_1, q_2)$ l'état de $\Delta = \Delta_1 \otimes \Delta_2$. Alors, on a $q_1 = q \downarrow \{Mod_{j1}, \ldots, Mod_{jp}, Obs_{j1}, \ldots, Obs_{jp}\}$. On peut également noter $q_1 = q \downarrow \{\Delta_1\}$ puisque $q_1 \in Q_1$ où Q_1 est l'ensemble des états de Δ_1. La projection permet ainsi de connaître l'état d'un sous-système du composant ou de projeter un état de diagnostic sur un sous-système.

Théorème 5.1.
Soit A_1 et A_2 deux automates. Soit I un ensemble d'états. Soit $A = (A_1 \otimes A_2)[I]$. Alors $A = (A_1[I \downarrow \{A_1\}] \otimes A_2[I \downarrow \{A_2\}])\ [I]$.

Démonstration. On note $A' = (A_1[I \downarrow \{A_1\}] \otimes A_2[I \downarrow \{A_2\}])\ [I]$.

Considérons une trajectoire $\text{chem} = q_0 \xrightarrow{l_1} \cdots \xrightarrow{l_k} q_k$ de A telle que $\forall i \in \{1, \ldots, k\}$, $q_i = (q_{i1}, q_{i2})$. Alors, puisque A est restreint sur I, $q_0 \in I$.

On a donc deux trajectoires $\text{chem}_j = q_{0j} \xrightarrow{l_{1j}} \cdots \xrightarrow{l_{kj}} q_{kj}$ (pour $j = \{1, 2\}$) telles que chem_j est une trajectoire de A_j. Or, $q_{0j} \in (I \downarrow A_j)$ puisque $(q_{01}, q_{02}) \in I$. Donc, chem_j est une trajectoire de $A_j[I \downarrow A_j]$.

Ainsi, chem est une trajectoire de $(A_1[I \downarrow A_1] \otimes A_2[I \downarrow A_2])$. Et donc, chem est une trajectoire de A'.

De même, toute trajectoire de A' est une trajectoire de A puisque la construction de A' comporte plus de restrictions que celle de A. Donc, $A = A'$. □

Ce théorème indique que si on fait un calcul intermédiaire lors du calcul de $A = (A_{j_1} \otimes \cdots \otimes A_{j_p} \otimes A_{j'_1} \otimes \cdots \otimes A_{j'_{p'}})[I]$, alors il est possible d'appliquer une restriction sur ces calculs intermédiaires et réduire ainsi la taille des automates.

Nous proposons d'utiliser ce théorème dans le cadre du calcul décentralisé du diagnostic incrémental.

Définition 5.6 (Diagnostic incrémental de sous-système)**.**
Soit F^{i-1} l'ensemble des états finaux de Δ_{i-1} Soit ss un sous-système et Δ^i_{ss} le morceau de diagnostic calculé grâce au diagnostic par morceau du sous-système ss. Le diagnostic incrémental du sous-système *ss noté $\Delta^i_{[ss]}$ est défini de la manière suivante : $\Delta^i_{[ss]} = \Delta^i_{ss}[F^{i-1} \downarrow \{\Delta^i_{ss}\}]$.*

Le diagnostic incrémental de sous-système consiste à utiliser l'ensemble des états finaux de la précédente fenêtre projeté sur le sous-système qui nous intéresse pour

restreindre le diagnostic du sous-système. Notons que la définition concerne également les sous-systèmes constitués d'un seul composant. Enfin, remarquons que si ss est le système complet, alors $\Delta^i_{[ss]}$ est le diagnostic incrémental du système pour la fenêtre i : $\Delta^i_{[\Gamma]} = \Delta^i_{\odot}$.

Résultat 5.2.
Soit F^{i-1} l'ensemble des états finaux de la fenêtre précédente. Soit ss un sous-système et Δ^i_{ss} le morceau de diagnostic calculé grâce au diagnostic par morceau du sous-système ss. Soit $ss_1, \ldots, ss_j$ une partition du sous-système ss.
Alors, on a : $\Delta^i_{[ss]} = (\Delta^i_{[ss_1]} \otimes \cdots \otimes \Delta^i_{[ss_j]})[F^{i-1} \downarrow \{\Delta^i_{ss}\}]$.

Démonstration. Ceci est une application immédiate du Théorème 5.1. □

Ce résultat est très intéressant puisqu'il indique qu'il est possible de raffiner localement le diagnostic des sous-systèmes avant de fusionner les diagnostics et d'appliquer le raffinement.

La figure 5.3 montre ce résultat sur un exemple de système comprenant quatre composants. Dans cet exemple, on construit le diagnostic local pour chaque composant puis on effectue un raffinement[1]. La synchronisation du diagnostic du premier composant avec le diagnostic du deuxième composant donne un diagnostic raffiné sur l'ensemble $I_A = I_1 \times I_2$ où I_1 et I_2 sont les ensembles d'états sur lesquels les diagnostics des deux premiers composants ont été restreints. Ensuite, il est possible de raffiner à nouveau le diagnostic sur un ensemble $I_{1,2} \subseteq I_1 \times I_2$. Il est possible de continuer jusqu'à obtenir le diagnostic du système.

5.3 Diagnostic décentralisé par chaînes d'automates

Le diagnostic décentralisé par chaîne d'automates consiste à effectuer des diagnostics décentralisés sur de petites périodes et d'utiliser le fait que les périodes soient petites pour bénéficier du diagnostic décentralisé. Nous commençons par deux exemples. Le premier permet de comprendre le principe de la méthode, le second permet de mettre en valeur les difficultés de la tâche. Nous montrons ensuite comment résoudre ce problème et nous donnons quelques expérimentations montrant les avantages de cette méthode.

5.3.1 Principe

Considérons par exemple la figure 5.4. Cet exemple montre deux diagnostics locaux pour deux composants. L'unique événement de synchronisation des deux diagnostics est l'événement c. La synchronisation des deux diagnostics produit l'automate présenté à la figure 5.5. On voit que l'automate produit comporte beaucoup plus d'états et de transitions que l'automate d'origine. La synchronisation apporte cependant une information puisqu'elle permet de supprimer l'hypothèse de l'occurrence des événements b_6 et b_7.

[1] En fait, la synchronisation et le raffinement sont simultanés.

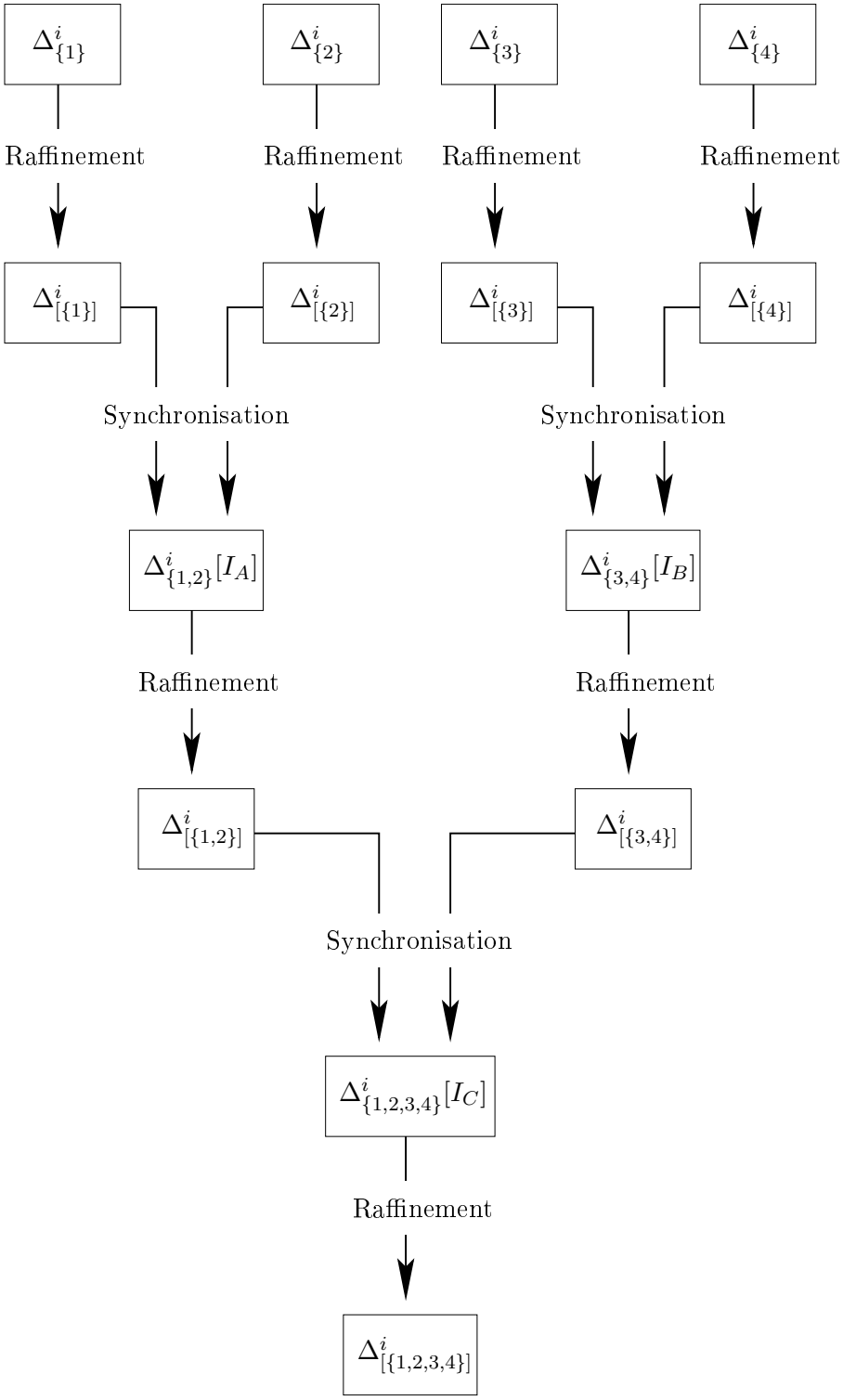

FIG. 5.3 – Principe du raffinement local

Remarquons que de manière générale, il n'est pas évident de raisonner sur des sous-systèmes aux comportements inter-dépendants à l'aide d'automates non synchronisés.

Notre but est d'obtenir un diagnostic de la forme présenté à la figure 5.6. Ici, le diagnostic est découpé en trois morceaux. Le premier morceau comprend les événements a_1, a_2, a_3, b_1, b_2 et b_3. Puisque les deux sous-systèmes ont un comportement indépendant pendant la période considérée, il est inutile d'effectuer la fusion des diagnostics. La deuxième période ne comprend que l'événement de synchronisation c. La synchronisation est alors effectuée. Enfin, la dernière période comprend les observations a_5, b_5, b_6 et b_7. Puisque l'état final du système de la précédente fenêtre était celui après l'occurrence de c, les événements b_6 et b_7 disparaissent. Il est par ailleurs inutile de synchroniser les deux diagnostics locaux. On voit que considérer des périodes plus petites permet de découvrir des périodes d'indépendance entre sous-systèmes et de profiter du diagnostic décentralisé.

5.3.2 Difficultés

La synchronisation incrémentale nécessite pour le raffinement le calcul pour chaque morceau de diagnostic de l'ensemble des états finaux. Ce calcul et son utilisation ne sont pas triviaux dans le cas décentralisé à cause de l'hétérogénéité des sous-systèmes d'un morceau de diagnostic au morceau suivant. Considérons l'exemple de la figure 5.7 (les étiquettes de transition ont été omises dans cet exemple). Dans cet exemple, les états du diagnostic sont étiquetés par l'état de deux composants. O représente l'état *ok* et P l'état *en panne*. L'état du second composant est primé. À l'issue du premier morceau, on est sûr qu'un des composants est en panne (sans savoir lequel : états OP' et PO'). Considérons que dans une fenêtre temporelle suivante, les deux composants évoluent indépendamment sans changer d'état. Dans le diagnostic local, l'état des composants est *ok* ou *en panne*. On voit qu'il n'est pas nécessaire de fusionner les diagnostics puisque cela n'apporte aucune information. Au début de la troisième fenêtre, il est nécessaire de retourner à la première fenêtre pour savoir que les deux composants du système ne sont pas dans l'état *ok*. Formellement, nous avons $F_1^2 = \{O, P\}$ (l'ensemble des états finaux du premier composant à l'issue du deuxième morceau), $F_2^2 = \{O', P'\}$ (l'ensemble des états finaux du second composant à l'issue du deuxième morceau) et $F^2 = \{(O, P'), (O', P)\}$ (l'ensemble des états finaux du système à l'issue du deuxième morceau) avec $F^2 \neq F_1^2 \times F_2^2$.

Le calcul devient particulièrement ardu lorsque les sous-systèmes indépendants varient d'un morceau de diagnostic à l'autre. Il nous semble donc nécessaire de calculer explicitement l'ensemble des états finaux du système. Nous proposons donc de calculer pour chaque morceau de diagnostic la liste des états finaux.

5.3.3 Résolution

La difficulté du diagnostic décentralisé par calcul incrémental est de travailler avec l'ensemble des états finaux lorsque celui-ci n'est pas explicitement calculé. Nous proposons donc de le calculer explicitement.

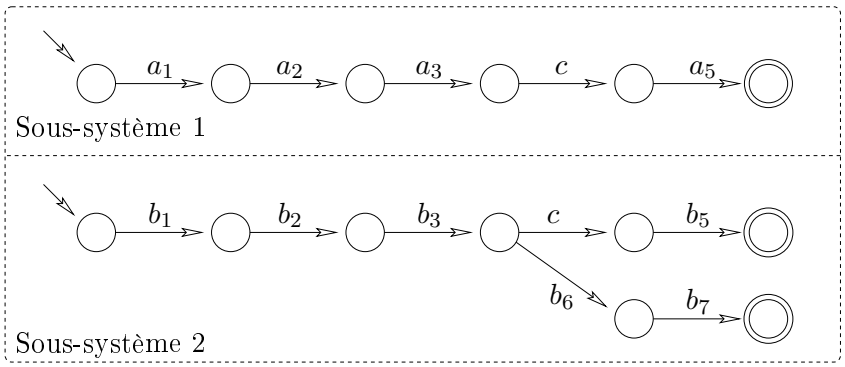

FIG. 5.4 – Exemple de deux diagnostics locaux

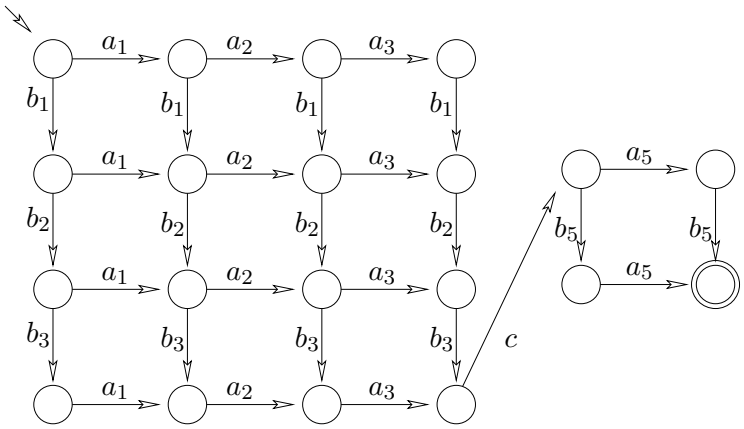

FIG. 5.5 – Synchronisation des diagnostics de la figure 5.4

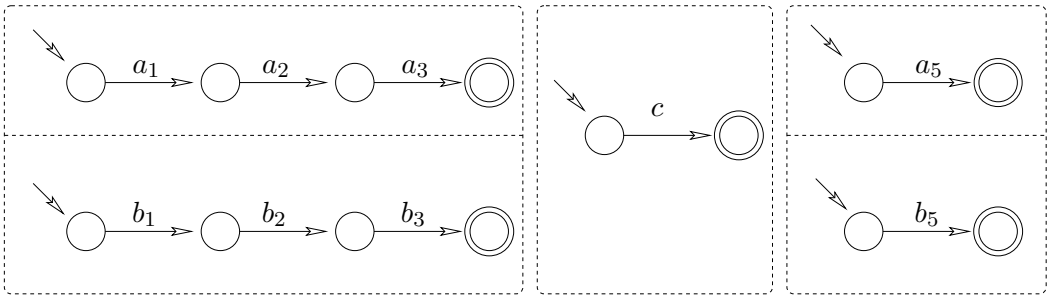

FIG. 5.6 – Principe du diagnostic décentralisé avec le diagnostic incrémental (pour l'exemple de la figure 5.4)

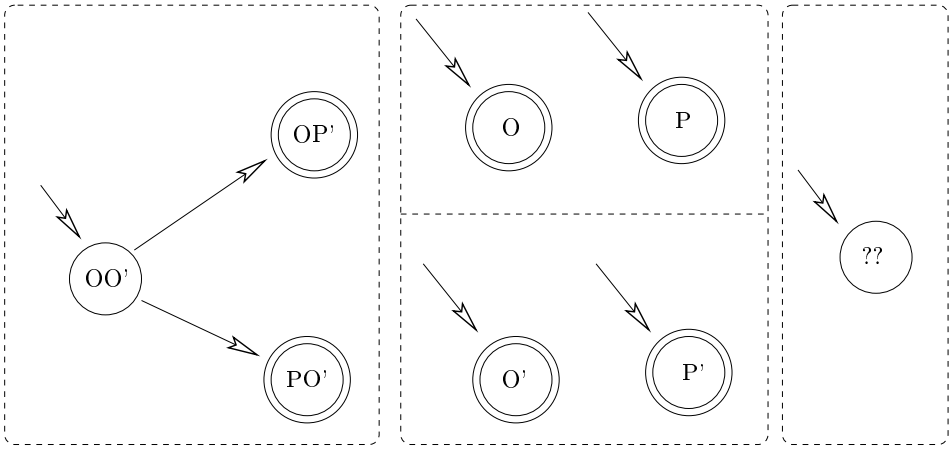

FIG. 5.7 – Quels sont les états finaux du deuxième morceau ?

On note $q \rightsquigarrow_A q'$ la propriété indiquant qu'il existe une trajectoire de q à q' sur l'automate A.

Résultat 5.3.
Soit F^{i-1} l'ensemble des états finaux du morceau $i-1$. Soit $ss_1, \ldots, ss_k$ une partition du système et $\Delta^i_{ss_1}, \ldots, \Delta^i_{ss_k}$ les diagnostics de chacun de ces sous-systèmes pour le morceau i tels que ces diagnostics sont deux à deux indépendants. Soit F l'ensemble des états défini par :

$\{q' \mid \exists q \in F^{i-1},\ \forall j \in \{1, \ldots, k\},\ (q \downarrow \{\Delta^i_{ss_j}\}) \rightsquigarrow_{\Delta^i_{ss_j}} (q' \downarrow \{\Delta^i_{ss_j}\})\}$.

Alors F est l'ensemble des états finaux du morceau Δ^i.

Nous avons donc montré comment calculer l'ensemble des états finaux de la fenêtre.

5.3.4 Expérimentation

Nous présentons dans cette partie une expérimentation du diagnostic incrémental décentralisé.

5.3.4.1 Système étudié

Le système considéré est un réseau de composants interconnectés comme présenté dans la figure 5.8. Deux composants sont dits voisins s'il existe une connexion entre ceux-ci.

Chacun des composants a le même fonctionnement. Lorsqu'une panne (*f*, pour *faute*) a lieu sur un composant, celui-ci demande à tous ses voisins de redémarrer (*r!*, pour *redémarre !*) et essaie de redémarrer. Lorsqu'un composant vient de subir une panne et reçoit un message de redémarrage (*r?*), il ignore le message. En revanche, s'il est en train de redémarrer, il redémarre à nouveau. Les événements de redémarrage sont *fdf* et *fdr* pour *fin de faute* et *fin de redémarrage*. Le début et la fin de redémarrage sont

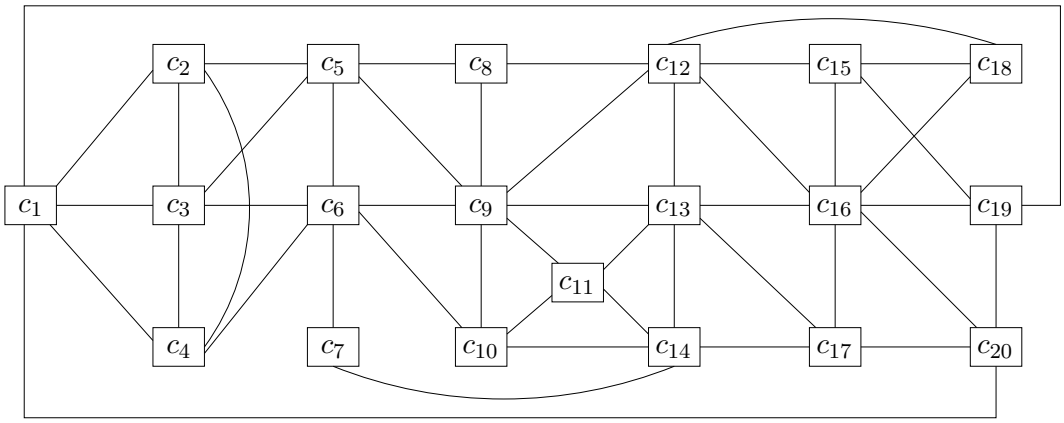

FIG. 5.8 – Topologie du système

observables (*JR* et *JSDR* pour *Je redémarre* et *Je suis de retour*). La figure 5.9 présente le modèle de manière simplifiée d'un composant.

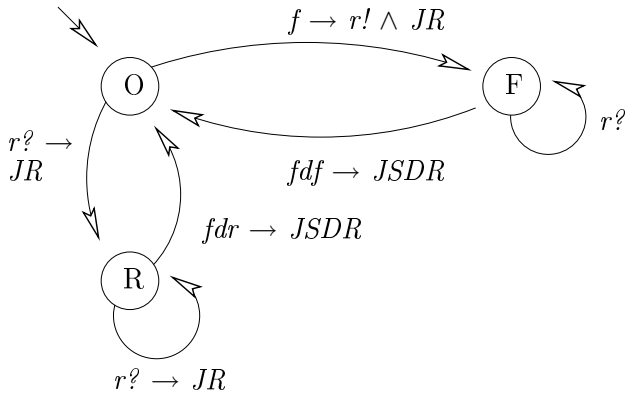

FIG. 5.9 – Modèle générique d'un composant

Les états O, F et R correspondent respectivement à *OK*, *Fautif* et en *Redémarrage*. Sur l'étiquette de transition, l'événement avant la flèche est l'événement déclencheur et les événements après sont les observations et les messages envoyés aux autres composants. L'événement $r?$ (resp. $r!$) indique la réception (resp. l'émission) du message *redémarre* depuis (resp. vers) un composant connexe. Cela signifie que $r!$ doit être remplacé par r_i_j où i est le composant modélisé et j tout composant connexe. De plus chaque transition comprenant $r?$ doit être remplacée par k transitions où $r?$ est remplacé par r_j_i (k est le nombre de composants connexes c_j du composant c_i). L'unique événement de faute est f (remplacé par f_i). Les observables sont JR (JR_i)

et *JSDR* (*JSDR_i*). Les événements de fin de redémarrage doivent également comporter un indice différent pour chaque composant.

La difficulté du diagnostic réside dans le fait que lorsqu'un composant tombe en panne, ou lorsqu'un de ses voisins tombe en panne, le composant émet les mêmes observations (*JR* suivi de *JSDR*). Aussi, il est nécessaire de raisonner de manière globale pour déterminer quel composant a subi la panne.

Si on cherche à construire le modèle du système, on obtient $|\{\mathrm{O},\mathrm{F},\mathrm{R}\}^{20}-1| \approx 3,4.10^9$ états[2]. Le modèle décentralisé ne comporte quant à lui que $3 \times 20 = 60$ états. Il est donc nécessaire d'utiliser le modèle décentralisé et un calcul décentralisé du diagnostic.

5.3.4.2 Mode opératoire

Les observations locales à chaque composant sont ordonnées et on y insère les *tics* d'horloge Υ_i (voir [GCL05b]) correspondant au passage d'une fenêtre temporelle à la fenêtre suivante. L'automate des observations locales est présenté figure 5.10.

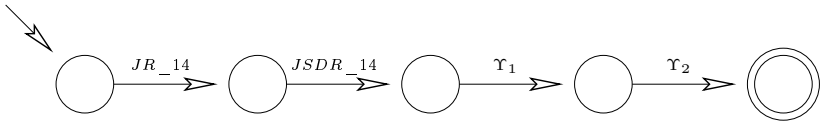

FIG. 5.10 – Automate des observations locales pour le composant numéro 14

Les tics d'horloge se passent simultanément sur tous les composants. Il convient donc de les synchroniser lors de la fusion des diagnostics. Cependant, s'il n'y a pas d'autres événements de synchronisation, leur synchronisation lors de la fusion des diagnostics apporte comme seule information l'ordre des pannes (et ne discrimine pas la localisation des pannes). Aussi, pour déterminer si deux sous-systèmes ont un comportement indépendant, on ne considère pas les événements Υ_i.

Les morceaux d'automates de la chaîne d'observations pour chaque composant sont découpés grâce à Υ_i. Ainsi, l'automate Obs_{14}^1 comporte trois états, reliés par deux transitions étiquetées par *JR_*14 et *JSDR_*14.

5.3.4.3 Expérimentation

La période à diagnotiquer comprend deux fenêtres temporelles. Durant la première fenêtre, les composants 3 et 11 tombent en panne forçant leurs voisins à redémarrer (19, 20, 2, 3 4 d'une part, 9, 10, 13, 14 d'autre part) puis reviennent en mode de fonctionnement normal, ainsi que leurs voisins. Durant la seconde période, le composant 6 tombe en panne et lui et ses voisins (3, 4, 5, 7, 9 et 10) redémarrent avec succès.

Avec la méthode de diagnostic décentralisé classique, on voit qu'on ne peut pas diviser le sous-système constitué des 14 composants cité ci-dessus en sous-systèmes

[2]L'état *(R,R,...,R)* n'est pas accessible.

indépendants. Leurs diagnostics locaux doivent donc être fusionnés. On obtient ainsi 7 diagnostics de sous-systèmes, dont 6 diagnostics locaux à un composant. Le dernier diagnostic est un automate comprenant 2 275 états et 179 012 transitions.

La méthode utilisant les chaînes d'automates avec le diagnostic décentralisé produit 25 automates. Ces automates totalisent 274 états et 2,963 transitions. Le nombre d'états a a donc été divisé par 8 et le nombre de transitions par 60.

5.4 Conclusion

Dans cette section, nous avons présenté les approches décentralisées, à savoir le calcul décentralisé du diagnostic et le diagnostic décentralisé.

Nous avons montré comment adapter les techniques de diagnostic par chaînes d'automates au calcul décentralisé du diagnostic pour profiter à la fois des avantages du diagnostic par chaînes d'automates (incrémentalité, calcul en-ligne) et des avantages du calcul décentralisé (efficacité du calcul pour les systèmes de grande taille).

D'autre part, nous avons montré qu'il était possible d'effectuer un diagnostic décentralisé par morceaux. Le diagnostic décentralisé permet de représenter de manière efficace le comportement d'un système par sous-systèmes indépendants. Nous avons montré que le découpage de la période de diagnostic en morceaux permet de découvrir plus de comportements indépendants, et donc de rendre plus efficaces les techniques de diagnostic décentralisé.

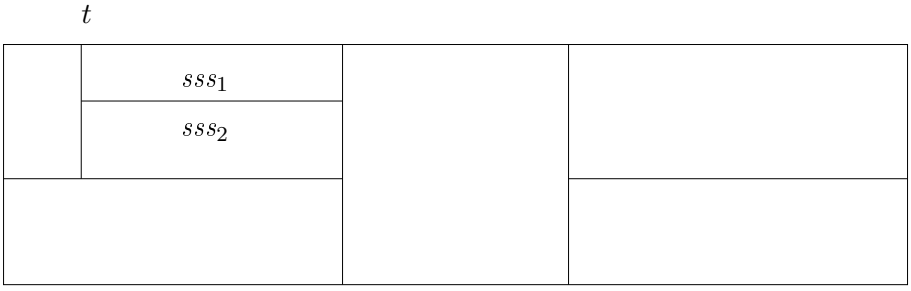

FIG. 5.11 – Imbrication du découpage et du décentralisé

Une extension possible de ces résultats serait de permettre une imbrication du découpage et de la décentralisation comme représenté de manière abstraite sur la figure 5.11. Dans cet exemple, le rectangle englobant représente le diagnostic global. Les traits verticaux correspondent au découpage du diagnostic et les traits horizontaux au diagnostic décentralisé. Ainsi, le diagnostic est découpé en trois fenêtres qui permettent parfois de produire des diagnostics de sous-système. Le diagnostic du premier sous-système lors de la première fenêtre temporelle peut à nouveau être découpé en sous-fenêtres et faire apparaître de nouveaux comportements indépendants pendant ces sous-fenêtres.

Chapitre 6

Calcul du diagnostic de systèmes reconfigurables par chaînes hétérogènes

Dans ce chapitre, nous considérons le diagnostic de systèmes reconfigurables. Nous laissons de côté les problématiques de calcul incrémental ou décentralisé, ainsi que de diagnostic décentralisé.

La reconfiguration est une opération modifiant le comportement du système, et donc son modèle. Nous proposons d'utiliser les chaînes d'automates pour le diagnostic. Dans ce contexte, le passage d'un morceau d'automate à un autre correspond à l'application d'une reconfiguration. Cependant, les espaces d'états sont différents avant et après l'application de la reconfiguration. Aussi, une extension des chaînes d'automates, appelée *chaîne hétérogène* est proposée.

Dans ce chapitre, nous donnons d'abord la définition de la reconfiguration, et comment celle-ci est modélisée. Puis, nous étendons la définition de chaînes d'automates et nous appliquons ces résultats pour le calcul du diagnostic.

6.1 Définition et modélisation de la reconfiguration

Nous avons donné en 1.5 la définition de reconfiguration suivante : *étant donné un modèle du système, une* reconfiguration *du système est une opération effectuée sur le système rendant son modèle obsolète.* Nous reprenons ici cette définition. Une reconfiguration est noté $\mathcal{R}$.

Soit *Mod* le modèle du système. La reconfiguration rend ce modèle obsolète ; on note $\mathcal{R}(Mod)$ le modèle du système après la reconfiguration.

Nous considérons tout d'abord l'hypothèse suivante :

Hypothèse 6.1.
Une reconfiguration est instantanée.

Si on considère une reconfiguration étalée sur le temps, alors nous considérons que cette reconfiguration se fait en plusieurs étapes séparées. Par exemple, si on considère le remplacement d'un composant du système, il faut généralement supprimer les connexions entre le composant et le reste du système, retirer le composant puis mettre le nouveau composant à sa place avant de finalement rétablir les connexions entre le reste du système et le nouveau composant. Cette procédure peut même être plus longue. Il y a deux manières de modéliser cette reconfiguration : soit on considère qu'elle a été instantanée, soit on considère qu'il y a eu plusieurs reconfigurations, la première ayant été la déconnexion, la deuxième le retrait du composant, *etc.*

Nous considérons que la reconfiguration est commandée par un opérateur extérieur et qu'aucune action n'est effectuée en même temps. D'autre part, nous considérons qu'une reconfiguration ne génère aucun événement de manière synchrone. Ainsi, puisqu'une reconfiguration est instantanée, il est impossible qu'un événement extérieur ait lieu en même temps que la reconfiguration.

Hypothèse 6.2.
Aucun événement n'a lieu en même temps qu'une reconfiguration.

Prenons l'exemple d'un ordinateur auquel on connecte une clef usb. Cette action peut être considérée comme une reconfiguration. La plupart des systèmes d'exploitation sont capables de détecter immédiatement la connexion et de réagir en conséquence. Cependant, il existe un délai entre la reconfiguration et l'événement de détection du nouveau périphérique. Ainsi, les deux événements ne sont pas simultanés.

Considérons à nouveau l'hypothèse 6.1 et l'exemple du remplacement du composant. On imagine facilement que, selon le contexte, il peut se passer de nombreux événements dans le système pendant le remplacement du composant. Dans ce cas, on voit qu'il n'est pas possible de considérer que le remplacement soit considéré comme une seule reconfiguration. En réalité, il y a une première reconfiguration, puis le système évolue selon son modèle, et une deuxième reconfiguration a lieu, *etc.* En revanche, si on considère qu'on peut négliger les comportements du système pendant le remplacement du composant, alors on pourra considérer le remplacement comme une seule reconfiguration.

Les ensembles d'états du système avant et après la reconfiguration ne sont généralement pas les mêmes. C'est notamment le cas si on considère qu'on peut ajouter ou supprimer des composants. Formellement, on a $Mod^1 = (Q^1, E^1, T^1, I^1, F^1)$ le modèle avant la reconfiguration et $\mathcal{R}(Mod^1) = Mod^2 = (Q^2, E^2, T^2, I^2, F^2)$ le modèle après la reconfiguration. Alors $Q^1 \cap Q^2$ peut être vide.

Nous notons $\overset{\mathcal{R}}{\mapsto} \subseteq Q^1 \times Q^2$ la relation indiquant l'état possible du système après la reconfiguration. Ainsi, soit $q^1 \in Q^1$ et $Q = \{q^2 \mid q^1 \overset{\mathcal{R}}{\mapsto} q^2\}$. Alors, si le système était dans l'état q^1 avant la reconfiguration, il se trouve dans un état $q^2 \in Q$ à l'issue de la reconfiguration. Si $Q = \emptyset$, alors la reconfiguration est impossible (soit parce qu'elle est physiquement impossible, soit parce qu'un opérateur n'effectuera pas la reconfiguration $\mathcal{R}$ lorsque le système est dans l'état q^1). Remarquons que normalement, l'ensemble

d'états initiaux I^2 de Mod^2 est l'ensemble des états qui peuvent être atteints à l'issue de la reconfiguration $\{q' \mid \exists q,\ q \overset{\mathcal{R}}{\mapsto} q'\}$.

6.2 Définition des chaînes hétérogènes

Le diagnostic que nous calculons pour les systèmes reconfigurables se décompose en fenêtres temporelles comme pour le diagnostic dans le cadre non reconfigurable. Les reconfigurations ont lieu pendant le passage d'une fenêtre à une autre. Cependant, l'ensemble d'états n'est pas homogène d'une fenêtre à une autre. Ainsi, dans le cas non reconfigurable, l'état du système est le même après un changement de fenêtre (voir la troisième condition de la définition 3.4 de chemin sur une chaîne d'automates). Cependant, nous avons vu que l'état du système pouvait changer pendant une reconfiguration. Aussi, nous définissons une autre structure, proche des chaînes d'automates, appelée chaîne d'automates hétérogène.

Définition 6.1 (Chaîne d'automates hétérogène)**.**
Une chaîne d'automates hétérogène*, notée ξ_A, est une double séquence d'automates $(A^1, \ldots, A^n)$ où $A^i = (Q^i, E^i, T^i, I^i, F^i)$ et de fonctions partielles $(\overset{\mathcal{R}_1}{\mapsto}, \ldots, \overset{\mathcal{R}_{n-1}}{\mapsto})$ telle que $\overset{\mathcal{R}_i}{\mapsto}$ associe à certains états de F^i un ensemble d'états de I^{i+1}.*

Une chaîne d'automates hétérogènes est donc une double séquence $((A^1, \ldots, A^n), (\overset{\mathcal{R}_1}{\mapsto}, \ldots, \overset{\mathcal{R}_{n-1}}{\mapsto}))$. La séquence de fonctions partielles $\overset{\mathcal{R}_i}{\mapsto}$ est parfois sous-entendue. La fonction $\overset{\mathcal{R}_i}{\mapsto}$ est appelée *fonction de correspondance*. Un exemple de chaîne d'automates hétérogène est présenté à la figure 6.1. Les fonctions $\overset{\mathcal{R}_i}{\mapsto}$ sont représentées par les flèches en pointillé. Ainsi, $\overset{\mathcal{R}_1}{\mapsto} = \{(3,6),(4,8)\}$.

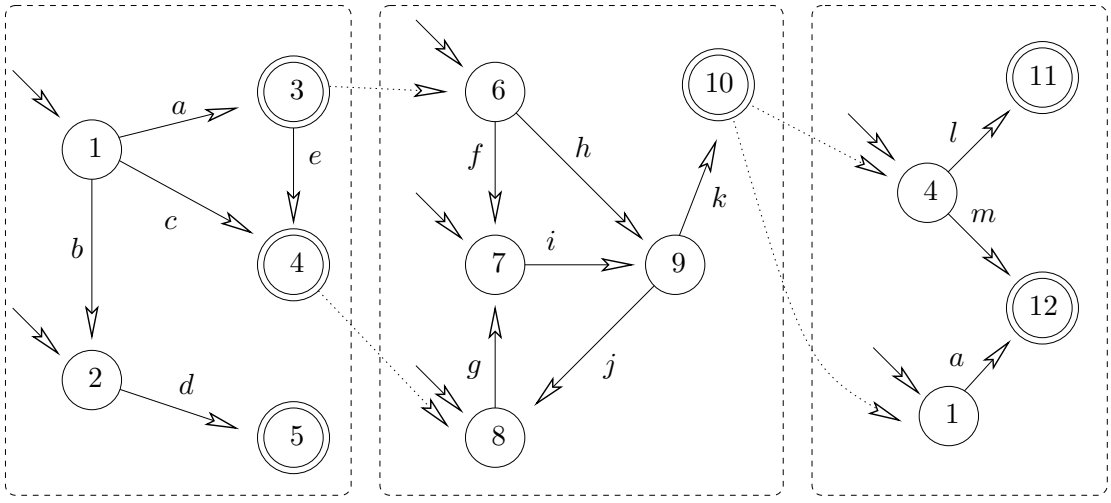

FIG. 6.1 – Exemple de chaîne d'automates hétérogène

On peut donc voir une chaîne d'automates comme une chaîne d'automates hétérogène telle que $\overset{\mathcal{R}_i}{\mapsto}$ est la fonction *identité*. Remarquons cependant que, contrairement aux chaînes d'automates, les chaînes hétérogènes ne peuvent pas être reconstruites.

Il est nécessaire de redéfinir les trajectoires pour les chaînes d'automates hétérogènes.

Définition 6.2 (Chemin d'une chaîne d'automates hétérogène)**.**
Soit $\xi_A = ((A^1, \ldots, A^n), (\overset{\mathcal{R}_1}{\mapsto}, \ldots, \overset{\mathcal{R}_{n-1}}{\mapsto}))$ *une chaîne d'automates hétérogène avec* $\forall i$, $A^i = (Q^i, E^i, T^i, I^i, F^i)$. *Soient* $i \in \{1, \ldots, n\}$ *et* $j \in \{i, \ldots, n\}$, *et soient* $(j - i + 1)$ *chemins* $\text{chem}^i, \ldots, \text{chem}^j$ *tels que* $\forall k \in \{i, \ldots, j\}$:
- $\text{chem}^k = ((q_0^k, \ldots, q_{mk}^k), (l_1^k, \ldots, l_{mk}^k))$ *est un chemin sur* A^k,
- $k \neq i \Rightarrow q_0^k \in I^k$,
- $k \neq j \Rightarrow q_{mk}^k \in F^k$,
- $k \neq j \Rightarrow q_{mk}^k \overset{\mathcal{R}_k}{\mapsto} q_0^{k+1}$.

Alors, $\text{chem} = ((q_0^i, \ldots, q_{mi}^i, q_0^{i+1}, \ldots, q_{mj}^j), (l_1^i, \ldots, l_{mi}^i, \mathcal{R}_i, l_1^{i+1}, \ldots, l_{mj}^j))$ *est un* chemin *sur la chaîne hétérogène* ξ_A.

La différence entre un chemin d'une chaîne d'automates et un chemin d'une chaîne d'automates hétérogène est que le premier ne répète pas l'état $q_0^{i+1} = q_{mi}^i$. Dans le second cas, puisque les états peuvent être différents, il est nécessaire de répéter l'état et de donner explicitement l'étiquette de transition $\mathcal{R}_i$. Sur l'exemple de la figure 3.1, $\text{chem} = ((3, 4, 8, 7), (e, \mathcal{R}_1, g))$ est un chemin.

Définition 6.3 (Trajectoire d'une chaîne d'automates hétérogène)**.**
Soit $\xi_A = ((A^1, \ldots, A^n), (\overset{\mathcal{R}_1}{\mapsto}, \ldots, \overset{\mathcal{R}_{n-1}}{\mapsto}))$ *une chaîne d'automates hétérogène avec* $\forall i$, $A^i = (Q^i, E^i, T^i, I^i, F^i)$. *Un chemin de* ξ_A $\text{chem} = ((q_0, \ldots, q_m), (l_1, \ldots, l_m))$ *est une* trajectoire *de* ξ_A *si* $q_0 \in I^1$ *et* $q_m \in I^n$ *et* $\forall i \in \{1, \ldots, n-1\}$, $\exists l_k = \mathcal{R}_i$.

Dans l'exemple de la figure 6.1, une trajectoire est $((1, 4, 8, 7, 9, 10, 4, 11), (c, \mathcal{R}_1, g, i, k, \mathcal{R}_2, l))$. Remarquons l'importance de la seconde condition de la définition nécessaire pour qu'un chemin soit une trajectoire. Sans cette condition, le chemin $((1, 12), (a))$ serait considéré comme une trajectoire.

Aucune trajectoire de la figure 6.1 ne passe par l'état 5. Ainsi, il est nécessaire de redéfinir l'opération de raffinement. Nous ne considérons ici que les I-raffinements, mais les F-raffinements peuvent être définis comme dans 3.3.1.

Définition 6.4 (I-raffinement)**.**
Soit $\xi_A = ((A^1, \ldots, A^n), (\overset{\mathcal{R}_1}{\mapsto}, \ldots, \overset{\mathcal{R}_{n-1}}{\mapsto}))$ *une chaîne d'automates hétérogène avec* $\forall i$, $A^i = (Q^i, E^i, T^i, I^i, F^i)$. *Le* I-raffinement *de* ξ_A *est une chaîne d'automates hétérogène* $\xi'_A = ((A'^1, \ldots, A'^m), (\overset{\mathcal{R}_1}{\mapsto}, \ldots, \overset{\mathcal{R}_{n-1}}{\mapsto}))$ *telle que* $\exists i \in \{2, \ldots, n\}$ *et* $\exists q \in I^i$ *tel que :*
- $\nexists q' \in F^{i-1}$ *avec* $q' \overset{\mathcal{R}_{i-1}}{\mapsto} q$,
- $\forall j \neq i$, $A'^j = A^j$ *et*
- $A'^i = (Q^i, E^i, T^i, I^i \setminus \{q\}, F^i)$.

La chaîne d'automates hétérogène est destinée à remplacer l'automate du modèle dans le diagnostic. Aussi, il est nécessaire de redéfinir l'opération de synchronisation entre un automate et une chaîne d'automate.

Il est possible de définir différentes synchronisations entre une chaîne d'automates et une chaîne d'automates hétérogène. Nous présentons ici la synchronisation simple puis nous étendons cette définition à la synchronisation incrémentale. Ces deux synchronisations sont à comparer aux synchronisations simple et incrémentale entre un automate et une chaîne d'automates.

La synchronisation entre une chaîne hétérogène $\xi_M = ((M^1, \ldots, M^n), (\overset{\mathcal{R}_1}{\mapsto}, \ldots, \overset{\mathcal{R}_{n-1}}{\mapsto}))$ et une chaîne d'automates $\mathcal{E}_A = (A^1, \ldots, A^n)$ produit une chaîne hétérogène (puisqu'à l'issue de cette synchronisation, on obtient une séquence d'automates dans des espaces d'états hétérogènes). L'automate i de la chaîne hétérogène est synchronisé avec l'automate i de la chaîne d'automates. Cela signifie que le passage de l'automate A^i à l'automate A^{i+1} se fait au moment du passage de M^i à M^{i+1} (pendant la reconfiguration $\mathcal{R}_i$). La relation de correspondance est mise à jour en disant qu'un état (q^i, q) correspond à un état (q^{i+1}, q) si q^i correspond à q^{i+1} et si q apparait à la fois dans A^i et dans A^{i+1}. Formellement :

Définition 6.5 (Synchronisation d'une chaîne hétérogène et une chaîne d'automates)**.**
Soit $\xi_M = ((M^1, \ldots, M^n), (\overset{\mathcal{R}_1}{\mapsto}, \ldots, \overset{\mathcal{R}_{n-1}}{\mapsto}))$ *une chaîne d'automates hétérogène avec* $\forall i$, $M^i = (Q^i_M, E^i_M, T^i_M, I^i_M, F^i_M)$. *Soit* $\mathcal{E}_A = (A^1, \ldots, A^n)$ *avec* $\forall i$, $A^i = (Q^i, E^i, T^i, I^i, F^i)$. *La* synchronisation *de la chaîne d'automates hétérogène* ξ_M *avec la chaîne d'automates* $\mathcal{E}_A$*, notée* $\xi_M \otimes \mathcal{E}_A$*, est la chaîne hétérogène* $\xi_{D\otimes} = ((D^1_\otimes, \ldots, D^n_\otimes), (\overset{\mathcal{R}_1}{\mapsto}_\otimes, \ldots, \overset{\mathcal{R}_{n-1}}{\mapsto}_\otimes))$ *où :*

- $\forall i$, $\overset{\mathcal{R}_i}{\mapsto}_\otimes = \{((q^i, q), (q^{i+1}, q)) \mid q^i \in Q^i_M \wedge q^{i+1} \in Q^{i+1}_M \wedge q \in Q^i \cap Q^{i+1} \wedge q^i \overset{\mathcal{R}_i}{\mapsto} q^{i+1}\}$, *et*
- $\forall i$, $D^i_\otimes = M^i \otimes A^i$.

Il est possible d'étendre cette définition comme nous l'avons fait pour la synchronisation incrémentale. La différence avec la synchronisation simple est que le résultat de $D^i_\odot$ est restreint sur l'ensemble des états q' qui ont un correspondant q dans $D^{i-1}_\odot$.

Définition 6.6 (Synchronisation incrémentale d'une chaîne hétérogène et une chaîne d'automates)**.**
Soit $\xi_M = ((M^1, \ldots, M^n), (\overset{\mathcal{R}_1}{\mapsto}, \ldots, \overset{\mathcal{R}_{n-1}}{\mapsto}))$ *une chaîne d'automates hétérogène avec* $\forall i$, $M^i = (Q^i_M, E^i_M, T^i_M, I^i_M, F^i_M)$. *Soit* $\mathcal{E}_A = (A^1, \ldots, A^n)$ *avec* $\forall i$, $A^i = (Q^i, E^i, T^i, I^i, F^i)$. *La* synchronisation incrémentale *de la chaîne hétérogène* ξ_M *avec la chaîne d'automates* $\mathcal{E}_A$*, notée* $\xi_M \odot \mathcal{E}_A$*, est la chaîne hétérogène* $\xi_{D\odot} = ((D^1_\odot, \ldots, D^n_\odot), (\overset{\mathcal{R}_1}{\mapsto}_\odot, \ldots, \overset{\mathcal{R}_{n-1}}{\mapsto}_\odot))$ *où :*

- $\forall i$, $\overset{\mathcal{R}_i}{\mapsto}_\odot = \{((q^i, q), (q^{i+1}, q)) \mid q^i \in Q^i_M \wedge q^{i+1} \in Q^{i+1}_M \wedge q \in Q^i \cap Q^{i+1} \wedge q^i \overset{\mathcal{R}_i}{\mapsto} q^{i+1}\}$,
- $D^1_\otimes = M^1 \otimes A^1$, *et*
- $\forall i \neq 1$, $D^i_\odot = (M^i \otimes A^i)[\{q' \mid \exists q, q \overset{\mathcal{R}_{i-1}}{\mapsto}_\odot q'\}]$.

6.3 Application au diagnostic de systèmes reconfigurables

Nous appliquons ces résultats pour le diagnostic de systèmes reconfigurables. Nous utilisons les découpages pour modifier le modèle du système. Nous montrons comment appliquer ce principe et donnons un exemple. Enfin, nous expliquons comment il est possible d'utiliser les chaînes d'automates hétérogènes quand il n'y a pas de reconfiguration.

6.3.1 Calcul du diagnostic

Le résultat peut s'appliquer directement au diagnostic. Nous considérons que toutes les reconfigurations sont connues.

Hypothèse 6.3.
On connait toutes les reconfigurations qui ont eu lieu sur le système.

Soit *Mod* le modèle initial du système et $\mathcal{R}_1, \ldots, \mathcal{R}_{n-1}$ la séquence de reconfigurations appliquées au système. Chaque reconfiguration $\mathcal{R}_i$ a eu lieu à la date t_i. Alors, on peut construire la chaîne hétérogène des modèles $\xi_{Mod} = ((Mod^1, \ldots, Mod^n), (\overset{\mathcal{R}_1}{\mapsto}, \ldots, \overset{\mathcal{R}_{n-1}}{\mapsto}))$.

Soit *Obs* l'automate des observations sur le système entre t_0 et t_n. Soit $\forall i$, $\mathcal{W}_i = [t_{i-1}, t_i]$. Soit $\mathcal{E}_{Obs}$ un découpage temporellement correct de *Obs* par rapport à $\mathcal{W}_1, \ldots, \mathcal{W}_n$. Le découpage doit être fait aux dates de reconfiguration, puisque comme nous l'avons vu, le passage d'un morceau d'observation à un autre est utilisé pour la reconfiguration.

Résultat 6.1.
Le diagnostic du système reconfigurable est $\xi_\Delta = \xi_{Mod} \odot \mathcal{E}_{Obs}$.

6.3.2 Exemple

Nous donnons ici un exemple. Le système est modélisé par la chaîne d'automates de la figure 6.2.

Nous considérons l'exemple d'un système vide tout d'abord. Ce système est donc modélisé par un état unique sans transition.

Puis, nous ajoutons deux composants c_1 et c_2 que nous connectons. Ces composants ont chacun un mode normal et un mode fautif qui peut être atteint par l'événement de panne f_1 ou f_2. La panne est permanente. Le modèle comporte donc 2×2 états. Lorsque l'événement extérieur *tick* a lieu, le composant c_1 émet le message *aok* (non observable) relayé par le composant c_2 qui émet alors *bok* qui est un événement observable. Si le second composant est en panne, il émet *bf* (message observable) ; et si le premier est en panne, il émet *af* (non observable) et le second composant émet *bf*. Le modèle du système est le deuxième automate de la chaîne hétérogène de la figure 6.2. Le superviseur reçoit l'observation *bf*.

L'opérateur effectue la reconfiguration suivante : il remplace le deuxième composant par un autre composant c_3, mais le deuxième composant est gardé déconnecté dans le

système. Le modèle du sous-système constitué de c_1 et c_3 est identique au modèle du système constitué de c_1 et c_2 à un renommage près. Le modèle est donné sur le troisième automate de la chaîne hétérogène de la figure 6.2. Le superviseur reçoit alors le message *cok*.

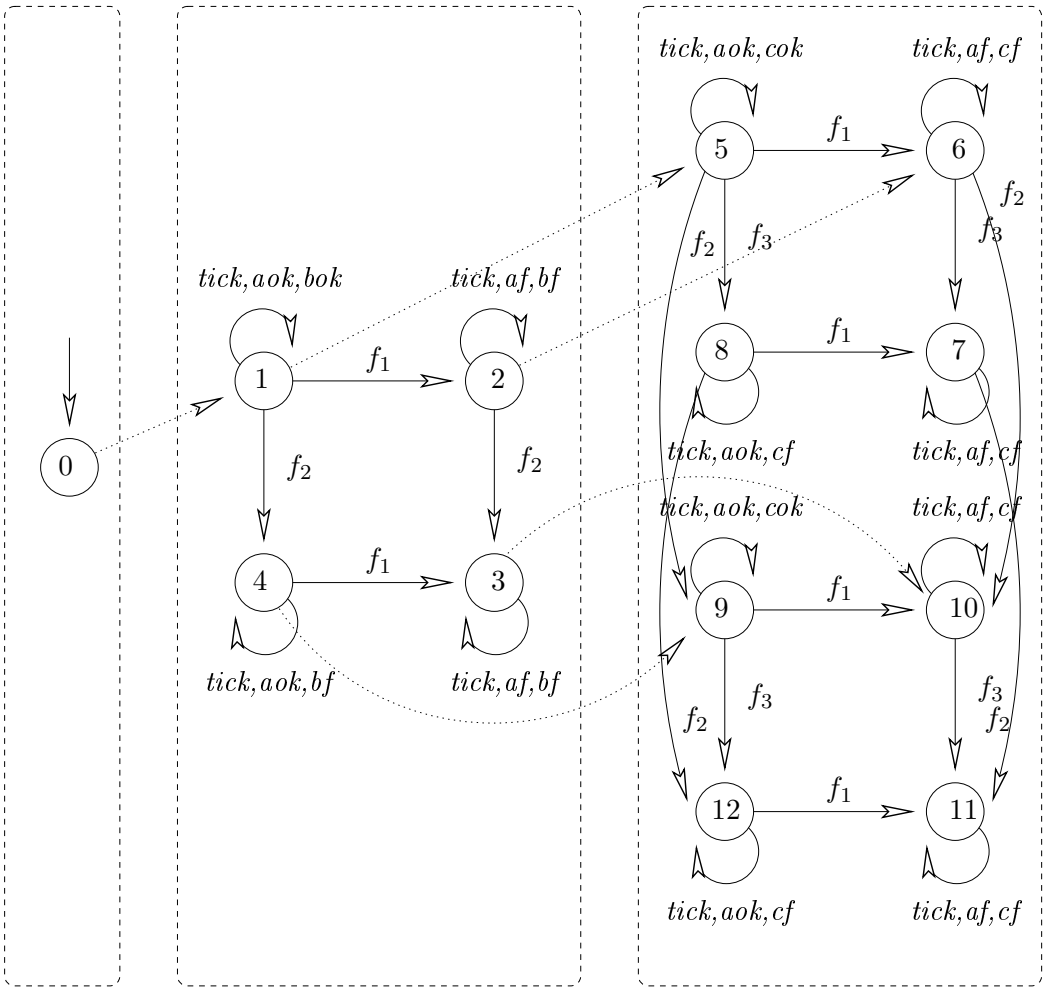

FIG. 6.2 – Chaîne hétérogène des modèles

La figure 6.2 donne la chaîne d'automates hétérogène modélisant le comportement du système. La chaîne des observations est présentée à la figure 6.3.

Le diagnostic est fourni à la figure 6.4. Remarquons que les états après $C9$ n'ont pas été représentés pour ne pas surcharger la figure, mais que l'automate n'a pas été simplifié pour montrer ce qui se passe. Les états qui doivent disparaitre par simplification sur le troisième automate sont représentés en grisé. Les états en grisé sur le deuxième automate ne sont pas supprimés par simplification mais par F-raffinement (si celui-ci est effectué). On peut interpréter l'automate de la manière suivante. Durant la deuxième

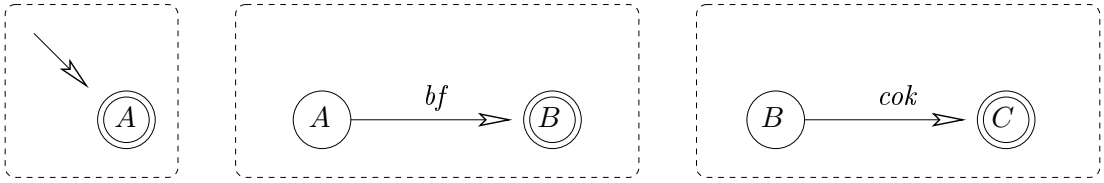

FIG. 6.3 – Chaîne des observations

fenêtre, l'observation *bf* ne permet pas de déterminer laquelle des deux pannes f_1 et f_2 a eu lieu. L'observation *cok* permet de déterminer que la panne f_1 n'a pas eu lieu. En effet, aucun des états 6 et 10 n'est suivi de l'observation *cok*. La seule trace possible sur le système est ($\mathcal{R}_1$, f_2, *{tick,aok,bf}*, $\mathcal{R}_2$, *{tick,aok,cok}*).

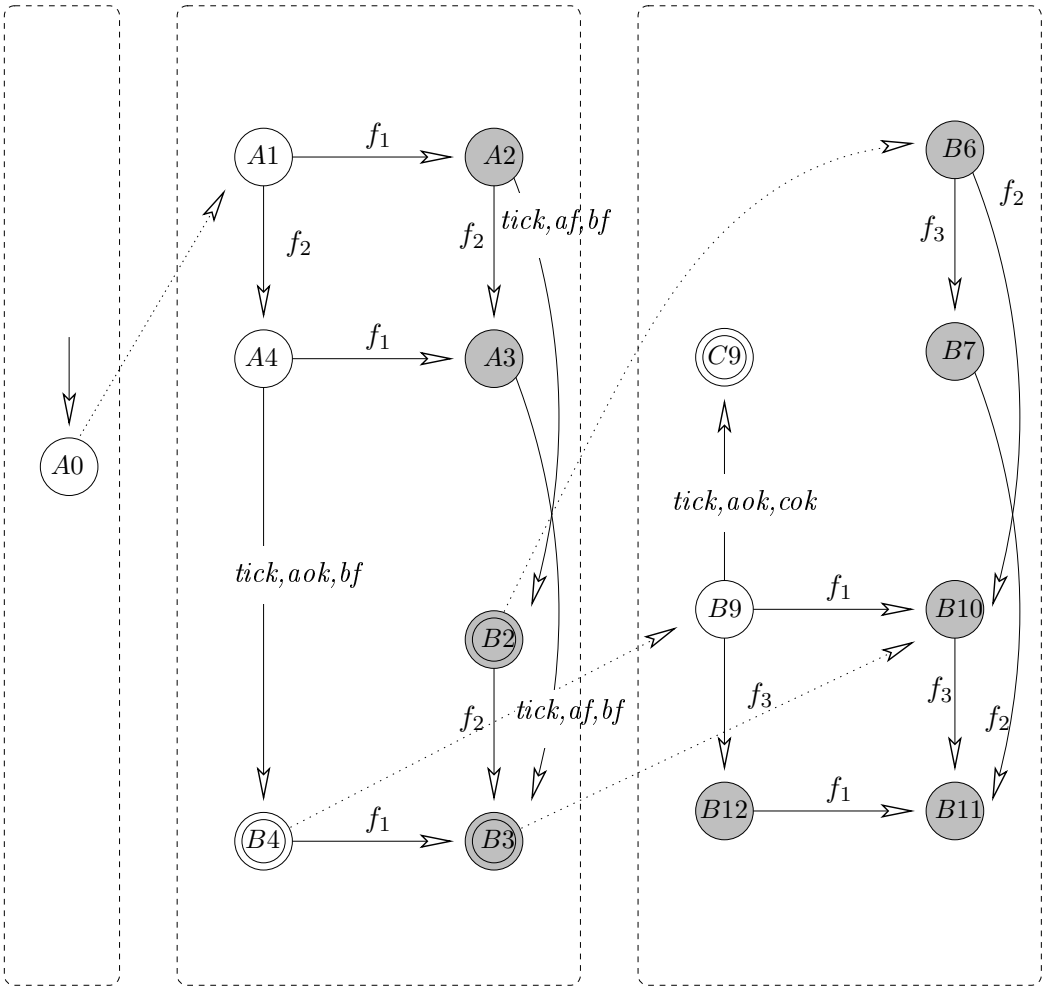

FIG. 6.4 – Le diagnostic du système

6.3.3 Découpages supplémentaires

Nous avons vu que le découpage des observations était intéressant pour le calcul du diagnostic de systèmes reconfigurables. Cependant, nous avons vu dans les précédents chapitres que le découpage pouvait être utile pour d'autres raisons (par exemple, pour permettre le diagnostic en-ligne, voir le chapitre 4). Dans notre présentation, une reconfiguration doit cependant être associée à chaque changement de fenêtre.

Nous proposons donc d'associer une reconfiguration *vide* pour chaque passage de fenêtre qui n'est pas associé à une reconfiguration. Le modèle n'est pas modifié par cette reconfiguration (à part que tous les états sont initiaux) ainsi que l'état du système.

Définition 6.7 (Reconfiguration vide)**.**
Une reconfiguration vide $\mathcal{R}_V$ *est la reconfiguration telle que* $\mathcal{R}_V(Mod) = Mod^{-}$ *et* $\overset{\mathcal{R}_V}{\mapsto} = Id$ *où* Id *est la fonction identité.*

On peut utiliser cette reconfiguration pour représenter le passage d'une fenêtre à une autre ne correspondant pas à une reconfiguration lors du diagnostic de système reconfigurable.

6.4 Conclusion

Dans ce chapitre, nous avons montré comment il était possible de diagnostiquer un système reconfigurable. Nous avons considéré que nous disposions du modèle du système avant et après la reconfiguration, et d'une description du changement d'état pendant la reconfiguration. Nous présentons au chapitre suivant une modélisation possible.

Le diagnostic de systèmes reconfigurables utilise une extension des chaînes d'automates appelée *chaîne hétérogène* qui permet la manipulation de chaîne d'automates dont les états ne sont pas dans le même espace.

Chapitre 7

Mise en œuvre

Dans les précédents chapitres, nous avons montré comment calculer le diagnostic du système de manière théorique. Durant cette thèse, nous avons développé un outil de diagnostic décentralisé de systèmes reconfigurables du nom de RAG$_e$. Dans ce chapitre, nous présentons cet outil.

Cette présentation nous permet de présenter une modélisation différente des systèmes. La modélisation que nous avons considérée dans les précédents chapitres était simplifiée de manière à simplifier les notations dans les définitions, théorèmes, résultats et autres preuves. Cependant, la notation proposée n'est pas très pratique, que ce soit d'un point de vue réutilisabilité ou dans le contexte de la reconfiguration. Ainsi, la relation $\overset{\mathcal{R}}{\mapsto}$ présentée dans le précédent chapitre est supposée connue mais aucun moyen de la calculer n'est donné. D'autre part, nous donnons ici les algorithmes utilisés dans RAG$_e$.

Le chapitre se découpe de la manière suivante. Dans une première section, nous donnons un exemple qui sert à illustrer les définitions suivantes. Puis, la modélisation choisie dans RAG$_e$ est présentée et nous montrons la correspondance de ce modèle avec celui utilisé dans les précédents chapitres. La section 7.3 montre l'extension de ce modèle pour autoriser les reconfigurations. Dans la section 7.4, nous donnons les formats des fichiers représentant les reconfigurations et les observations. Enfin, les algorithmes sont présentés en section 7.5.

7.1 Exemple

Nous reprenons l'exemple de la pompe et des tuyaux, présenté et utilisé dans divers articles [CPR00, CL01, GCL04]. Le système est composé d'une pompe et de tuyaux. Dans cet exemple, le système comporte une pompe et deux tuyaux, mais puisqu'on s'intéresse à des systèmes reconfigurables, il faut garder à l'esprit qu'on peut vouloir ajouter ou retirer des composants.

Le système est présenté sur la figure 7.1. Il y a trois composants (une pompe et deux tuyaux). Remarquons que les tuyaux sont identiques. La figure présente en plus trois configurations possibles pour le système.

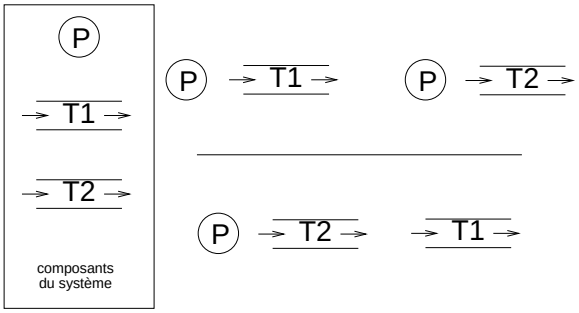

FIG. 7.1 – Le système pompe/tuyau

Le fonctionnement du système est le suivant : la pompe délivre un flux en sortie. Ce flux est *élevé*. En cas de panne, ce flux devient *faible* voire *nul* pour une panne grave. Une pompe ne peut pas être réparée.

En bon état, le tuyau renvoie en sortie un flux égal au flux d'entrée. En revanche, s'il est percé, le flux de sortie est inférieur au flux d'entrée. Un tuyau ne peut également pas être réparé.

Dans le cadre de la reconfiguration, on considère qu'une pompe ou un tuyau ne peuvent être déconnectés si un liquide s'écoule par le point à déconnecter ou à connecter.

Dans un premier temps, nous considérons que le système comprend une pompe connectée à un tuyau.

7.2 Modélisation

Cette section présente la modélisation décentralisée de système utilisée dans RAG_e. L'aspect *reconfiguration* est développé dans la section 7.3. Cette modélisation s'appuie fortement sur [PC05, LZ03b, RC02]. L'approche est une modélisation décentralisée comme [PC05], considérant une représentation explicite des ports comme pour [RC02]. Par rapport à [LZ03b], notre approche considère que les connexions sont synchrones.

7.2.1 Principe

Modéliser les systèmes de manière décentralisée revient à considérer un système comme présenté sur la figure 7.2. Le système est un ensemble de composants interconnectés. Pour modéliser le système, il suffit donc de donner le modèle de chaque type de composant, indiquer quelles sont les instances de chaque type de composant et enfin donner la liste des connexions entre les composants (topologie). Il peut y avoir plusieurs instances du même type de composant ; par exemple sur la figure 7.2, les composants 1 et 2 disposent du même modèle.

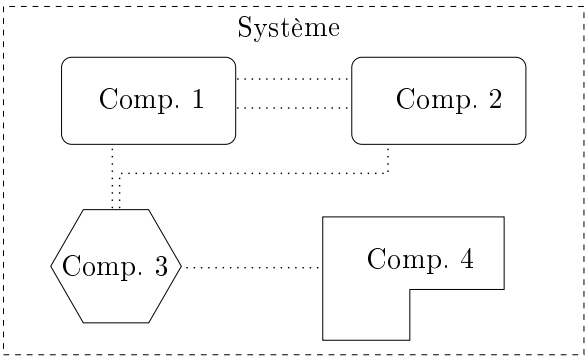

FIG. 7.2 – Principe du système décentralisé

7.2.2 Modèle d'un composant

Un *composant* est un élément (physique ou abstrait). Chaque composant dispose de *points de connexion* (parfois simplement appelés *points* ou *ports*). Un point permet de lier au travers d'une *connexion* le composant à un autre composant (c'est alors un point *interne*) ou à l'*environnement* (c'est-à-dire tout sauf le système, c'est alors un point *externe* ou *d'observations*).

Les *communications* (flots, messages, *etc.*) entre composants sont donc effectuées au travers des connexions. Par abus de langage, nous appelons *message* le contenu d'une communication. Si un message est envoyé depuis un composant vers un autre composant, le message est dit *interne*. Un message provenant de l'environnement est appelé message *exogène*. Enfin, les messages d'un composant vers l'environnement sont les messages *observables*. Il est généralement admis qu'un seul message peut être reçu par un composant à un instant donné. En conséquence, nous pouvons faire l'hypothèse suivante :

Hypothèse 7.1.
Chaque composant n'a qu'un seul point de connexion connecté avec l'environnement pouvant recevoir des messages. Ce point externe unique est noté p^{ext}.

Les observations sont émises sur un canal particulier appelé *point d'observations* p^{obs}.

Hypothèse 7.2.
Chaque composant dispose d'un point d'observations p^{obs}.

Un composant est considéré comme un système réactif. L'état du système n'évolue donc que sur réception d'un message. Comme généralement, on fait l'hypothèse suivante :

Hypothèse 7.3.
Un composant ne peut recevoir qu'un seul message (qu'il soit interne ou exogène) à un moment donné.

Le comportement d'un composant est décrit par un automate communicant.

Définition 7.1 (Modèle d'un composant)**.**
Le modèle d'un composant est décrit par un automate communicant $\Sigma = \langle Q, E, P, T, Q^o \rangle$ *où :*

- Q *est l'ensemble des états du composant,*
- E *est l'ensemble des messages qui peuvent être reçus ou émis par le composant,*
- P *est l'ensemble des points de connexion du composant,* p^{ext} *est le point externe,*
- $T \subset (Q \times (P \times E) \times (P \times E)^* \times Q)$ *est l'ensemble des transitions,*
- Q^o *est l'ensemble des états initiaux.*

Une transition $t = (q, (p, e), \{(p_1, e_1), \ldots, (p_k, e_k)\}, q')$ peut être comprise de la manière suivante : le composant est dans l'état q et reçoit le message e sur son point p. Alors le composant renvoie immédiatement les messages e_i ($i \in \{1, \ldots, k\}$) sur ses points p_i et passe dans l'état q'. Remarquons que les transitions ne sont pas forcément déterministes (*c'est-à-dire* qu'il peut y avoir $t^1 = (q, (p, e), \{(p_1^1, e_1^1), \ldots, (p_{k^1}^1, e_{k^1}^1)\}, q')$ et $t^2 = (q, (p, e), \{(p_1^2, e_1^2), \ldots, (p_{k^2}^2, e_{k^2}^2)\}, q')$ avec $t^1 \neq t^2$). Aucun message ne peut être émis sur le point externe p^{ext}, et aucun message ne peut être reçu du point d'observations.

Il est possible, sans perdre d'expressivité, de considérer que chaque point de connexion permet seulement la réception ou l'émission de message (il s'agit alors d'un point d'entrée ou de sortie). Pour ne pas alourdir la notation, nous avons laissé cette possibilité entre parenthèses.

Pour permettre la réutilisation des modèles, plusieurs composants peuvent avoir le même modèle. À chaque composant c, on associe son modèle par la fonction *LMod*, où $LMod(c)$ est une instance du modèle. La principale différence entre deux instances du même modèle est que les états et les points de connexion sont indicés par les composants.

Pour illustrer cette définition, les modèles de la pompe et du tuyau sont présentés sur les figures 7.3 et 7.4 (les transitions du modèle du tuyau sont présentée sur la table 7.1). Les points d'observations ont été mis entre parenthèses pour cet exemple.

Nous avons décidé de discrétiser les flux de liquide entre les composants. Pour cela, nous donnons trois valeurs possibles au flux : élevé (h pour *high*), faible (l pour *low*) et nul (z pour *zero*). La sémantique des états du modèle du tuyau est la suivante :

- 1 : tuyau non percé, flot d'entrée nul,
- 2 : tuyau non percé, flot d'entrée faible,
- 3 : tuyau non percé, flot d'entrée élevé,
- 4 : tuyau percé, flot d'entrée nul, sortie nulle,
- 5 : tuyau percé, flot d'entrée faible conduisant à une sortie nulle,
- 6 : tuyau percé, flot d'entrée élevé ou faible conduisant à une sortie non nulle.

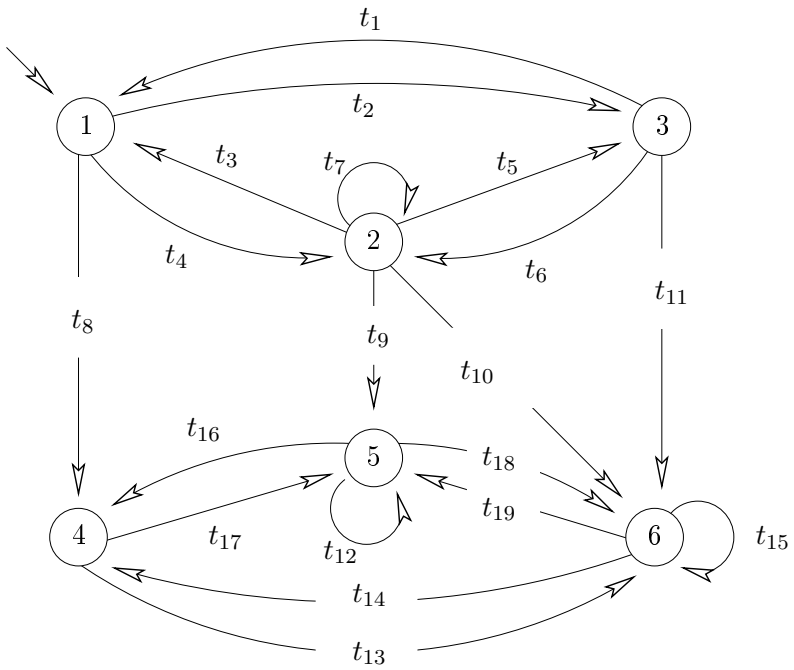

FIG. 7.3 – Modèle d'un tuyau

Transition	Étiquette
t_1	*in:z/{out:z}*
t_2	*in:h/{out:h}*
t_3	*in:z/{out:z}*
t_4	*in:l/{out:l}*
t_5	*in:h/{out:h}*
t_6	*in:l/{out:l}*
t_7	*in:l/{out:l}*
t_8	*ext:f/{}*
t_9	*ext:f/{out:z}*
t_{10}	*ext:f/{out:l}*
t_{11}	*ext:f/{out:l}*
t_{12}	*in:l/{}*
t_{13}	*in:l/{out:l}* et *in:h/{out:l}*
t_{14}	*in:z/{}*
t_{15}	*in:l/{out:l}* et *in:h/{out:l}*
t_{16}	*in:z/{}*
t_{17}	*in:l/{}*
t_{18}	*in:l/{out:l}* et *in:h/{out:l}*
t_{19}	*in:l/{}*

TAB. 7.1 – Table des étiquettes de transition de la figure 7.3

Notons que la discrétisation induit un non déterminisme sur le tuyau. Ainsi, lorsqu'un tuyau est percé et reçoit un flux d'entrée faible, il peut fournir en sortie soit un flux nul, soit un flux faible (voir par exemple les transitions t_9 et t_{10}). D'autre part, lorsque le flot d'entrée est élevé puis change, il passe nécessairement à faible ou nul. En revanche, un flot d'entrée faible peut changer mais rester à faible. On peut considérer l'exemple suivant correspondant à la transition t_{19} : le tuyau est percé et le flot d'entrée du tuyau est faible mais suffisant pour fournir un flot de sortie non nul ; puis, le flot d'entrée diminue (mais reste faible) et le flot de sortie passe à zéro.

La sémantique des états de la pompe est la suivante :

- 1 : la pompe fonctionne correctement mais est éteinte,
- 2 : la pompe fonctionne correctement et est allumée,
- 3 : la pompe souffre d'une panne légère et est allumée,
- 4 : la panne souffre d'une panne grave et est allumée,
- 5 : la panne souffre d'une panne légère et est éteinte,
- 6 : la panne souffre d'une panne grave et est éteinte.

Dans le modèle de la pompe, nous avons décidé de considérer qu'il existait deux types de panne : $f1$ correspondant à une panne légère et $f2$ correspondant à une panne grave. Nous avons considéré que les pannes ne peuvent avoir lieu que si la pompe est en marche. De plus, une fois que la panne $f2$ a eu lieu, aucune panne ne peut plus avoir lieu puisque la pompe est bloquée.

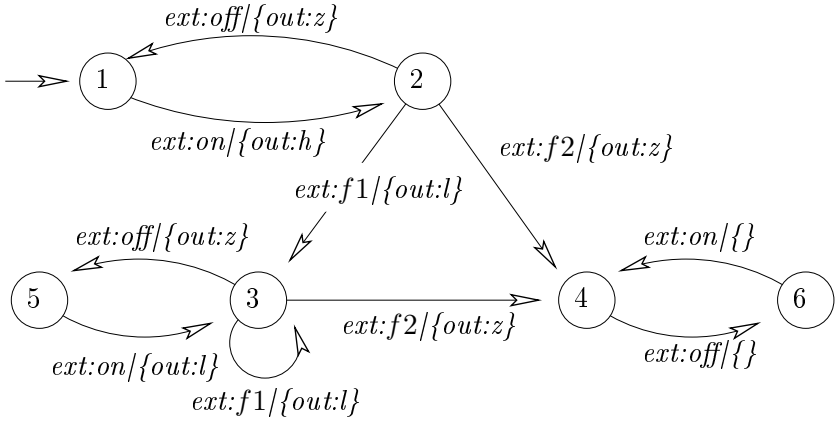

FIG. 7.4 – Modèle d'une pompe

7.2.3 Modèle de la topologie

Comme nous l'avons présenté, les composants sont modélisés par des systèmes à événements discrets. Ainsi, l'état du composant est modifié par la réception d'un message qui déclenche l'envoi de messages vers les autres composants. Il est donc clair que le comportement de chaque composant est contraint par leur manière de communiquer. Cette contrainte est appelée *synchronisation* et est décrite par un *ensemble de synchronisation*.

Définition 7.2 (Ensemble de synchronisation)**.**
Un ensemble de synchronisation*, noté* $|$*, entre les modèles de deux composants* $\Sigma_1 = \langle Q_1, E_1, P_1, T_1, Q_1^o \rangle$ *et* $\Sigma_2 = \langle Q_2, E_2, P_2, T_2, Q_2^o \rangle$ *est un sous-ensemble des paires de messages provenant des deux automates communicants :* $| \subseteq E_1 \times E_2$*.*

Un ensemble de synchronisation indique quel message est reçu lorsque tel message est envoyé. Il y a de nombreux intérêts à cette méthode. Tout d'abord, elle permet d'utiliser différents ensembles de messages pour les différents automates (et permettre ainsi une évolution dans la modélisation des composants). De plus, cette modélisation permet de représenter le fait que certains messages peuvent être interprétés de plusieurs manières par un composant. Par exemple, de nouveaux composants peuvent disposer d'un nouveau protocole. Ainsi, lors de l'envoi d'un nouveau type de message à un composant *obsolète*, ce composant pourra interpréter le message comme un autre type de message. Dans ce cas, on aura un ensemble de synchronisation de cette forme : {(ancien_message,ancien_message), (nouveau_message,ancien_message)}.

Dans l'exemple de la pompe et du tuyau, ainsi que dans la plupart des cas, on aura un ensemble de synchronisation égal à la fonction identité : $| = \{(h,h),(l,l),(z,z)\}$. Cet ensemble de synchronisation indique que le flot qui entre dans un composant est exactement celui qui sort du composant qui est connecté à celui-ci.

Une connexion peut maintenant se définir comme étant les deux points de connexion connectés, ainsi que l'ensemble de synchronisation associé.

Définition 7.3 (Connexion)**.**
Une connexion co est définie comme un tuple $((c_1,p_1),(c_2,p_2),|)$ *tel que :*

- $c_1 \neq c_2$,
- $\forall k \in \{1,2\}, \Sigma_k = \langle Q_k, E_k, P_k, T_k, Q_k^o \rangle = LMod(c_k),\ p_k \in P_k \wedge p_k \neq p_k^{\text{ext}} \wedge p_k \neq p_k^{\text{obs}}$,
- $|$ *est un ensemble de synchronisation entre* E_1 *et* E_2*.*

Il est impossible de connecter entre eux deux points de connexion du même composant. La connexion se fait entre deux points de connexion de deux composants avec un ensemble de synchronisation qui explique comment les messages sont transmis. Notons que le point externe et le point d'observations ne peuvent être connectés à un autre point de connexion.

Dans l'exemple de la pompe et du tuyau, la connexion se note $co = ((P,out),(T1,in),Id)$. Le point de connexion *out* de la pompe P est connecté au point *in* du tuyau $T1$. La fonction de synchronisation est la fonction identité.

L'ensemble des connexions est appelé la *topologie* (ou la *configuration*).

Définition 7.4 (Modèle de la topologie)**.**
Le modèle de la topologie *du système Top est un ensemble des connexions entre les composants du système tel qu'aucun point de connexion n'est connecté plus d'une fois :* $\forall co, co' \in Top,\ co = ((c_1,p_1),(c_2,p_2),|),\ co' = ((c'_1,p'_1),(c'_2,p'_2),|')$ *tel que* $co \neq co'$, $\forall i \in \{1,2\}, \forall j \in \{1,2\}, c_i = c'_j \Rightarrow p_i \neq p'_j$.

La topologie représente donc les connexions entre les composants du système. Notons que les connexions entre les composants et l'environnement ne sont pas explicitement indiquées (elles sont implicites).

On peut remarquer que ce modèle de la topologie est une forme simplifiée des modèles de liens proposés par Lamperti et Zanella [LZ03b].

7.2.4 Modèle décentralisé du système

Le modèle du système dépend du modèle de chacun des composants et de la façon qu'ont les composants d'interagir. Il convient cependant de faire certaines hypothèses sur le comportement du système. Il convient de remarquer que ces hypothèses sont des extensions directes de celles présentées dans [PC05]. La principale différence est l'addition d'une topologie explicite du système qui n'était présentée que de manière implicite dans [PC05]. Donner la topologie de manière explicite est indispensable quand on s'intéresse à des systèmes reconfigurables, puisque la topologie peut changer à tout moment.

7.2.4.1 Hypothèses simplificatrices

Le système est modélisé comme un système réactif, ce qui signifie qu'il évolue par l'occurrence de messages exogènes. Nous faisons donc l'hypothèse classique suivante :

Hypothèse 7.4.
Le système ne reçoit qu'un message externe à un moment donné.

La prochaine hypothèse indique qu'on s'intéresse à des systèmes synchrones. Notons qu'il est possible de représenter des comportements asynchrones en modélisant les connexions comme des composants comme montré dans la figure 7.5. La connexion est alors un composant qui a le comportement d'un tampon. Les messages émis par le composant 1 sont reçus par la connexion qui change d'état. Tant qu'elle a des messages non retransmis, la connexion peut envoyer un message au composant 2. De cette manière, il est possible de modéliser des comportements complexes (tampon de messages de taille *finie*, perte de messages).

Hypothèse 7.5.
Les communications entre les composants sont instantanées.

Comme nous l'avons indiqué, un composant est modélisé comme un système réactif, c'est-à-dire qu'il ne peut recevoir qu'un message à un instant donné. Les communications entre composants sont instantanées, et lorsqu'un composant reçoit un message, il peut à nouveau envoyer des messages à d'autres composants. Il convient donc de considérer l'hypothèse suivante :

Hypothèse 7.6.
Dans l'arbre des composants impliqués par la propagation des messages, un composant n'est présent qu'une seule fois.

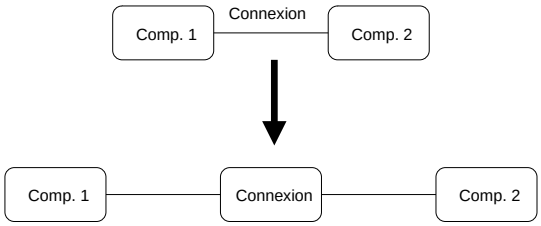

FIG. 7.5 – Modélisation asynchrone

Cela signifie qu'il n'y a pas de boucle de propagation, mais pas seulement. Par exemple, il ne peut pas y avoir le comportement suivant : le composant A envoie un message à chacun des deux composants B et C, qui à leur tour envoient un message au composant D, le tout de manière synchrone. En effet, dans cet exemple, le composant D est présent deux fois dans l'arbre de propagation.

7.2.4.2 Modèle décentralisé du système

Il est à présent possible de donner le modèle décentralisé du système.

Définition 7.5 (Modèle décentralisé du système)**.**
Le modèle décentralisé d'un système *est un n-uplet* $(Comp, LMod, Top)$ *où :*
- $Comp$ *est l'ensemble de composants du système,*
- $LMod$ *est une fonction qui donne le modèle de chaque composant,*
- Top *est la topologie du système.*

Le système est constitué d'un ensemble de composants. À chacun de ces composants, est associé un modèle local. Les connexions entre les composants sont explicitées dans la topologie.

Définition 7.6 (État du système)**.**
L'état q *du système* $(Comp, LMod, Top)$ *est une fonction qui associe à chacun de ses composants* $c \in Comp$ *un état du modèle* $LMod(c)$ *de ce composant.*

Par exemple, l'état du système constitué d'une pompe (éteinte et en parfait état) et d'un tuyau (non percé et ne recevant pas d'eau) est noté de la manière suivante : $q = (P \to 1, T1 \to 1)$. Lorsque les modèles sont différents (et les états des modèles sont disjoints) ou lorsque l'ordre des composants est implicitement connu, on peut noter $q = (q_1, \ldots, q_n)$ où q_i est l'état du i^{e} composant (ici, $q = (1, 1)$). Soit q, l'état du système, on note $q(c)$, l'état du composant c.

Remarquons que l'état du système ne comprend pas la topologie qui fait partie du modèle du système.

À la réception d'un message m sur le point externe p^{ext} du composant c, ce composant emprunte *une* transition $t = (q, (p, e), \{(p_1, e_1), \ldots, (p_k, e_k)\}, q')$ où q est l'état

courant du composant, $p = p^{\text{ext}}$ et $e = m$. Les messages e_i ($\forall i \in \{1, \ldots, k\}$) sont alors envoyés sur les points de connexion p_i. Si aucune connexion n'est associé au point de connexion, alors le message est perdu. En revanche, si une connexion $((c, p_i), (c', p'_i), |)$ est associée à p_i, alors un message m'_i est reçu par le composant c' sur son point de connexion p'_i avec $(m_i, m'_i) \in |$. S'il y a plus d'un message m'_i, alors un seul message est reçu par c'. S'il n'existe pas de message m'_i tel que $(m_i, m'_i) \in |$, alors la transition t n'a pas pu être franchie.

Le composant c' ayant reçu un message, il évolue également immédiatement. Bien évidement, le composant c a pu envoyer plusieurs messages en même temps, et ainsi, plusieurs composants évoluent de manière synchrone. Chaque composant recevant un message peut à son tour envoyer un ou plusieurs autres messages à d'autres composants. La contrainte est qu'aucun composant ne peut recevoir plusieurs messages à un instant donné. On note alors la transition effectuée sur le système de la façon suivante :

Définition 7.7 (Transition sur le système)**.**
Une transition t *sur le système* $(Comp, LMod, Top)$ *est une fonction partielle qui associe à certains composants* c *une transition* $t_c = (q_c, (p_c, e_c), \{(p_{ck}, e_{ck})\}, q'_c)$ *du modèle* $LMod(c)$ *de ce composant tel que :*

- $\exists! c,\ p_c = p^{\text{ext}}$,
- $\forall c',\ p_{c'} \neq p^{\text{ext}}, \exists c, \exists k$ *tel que* $\exists\ |$ *avec* $(e_{ck}, e') \in |$ *et* $co = ((c, p_{ck}), (c', p_{c'}), |) \in Top$,
- $\forall c, \forall k$ *tel que* $\exists c', \exists p', \exists\ |, \exists co = ((c, p_{ck}), (c', p'), |) \in Top$, *alors* $p_{c'} = p'$ *et* $(e_{ck}, e_{c'}) \in |$.

La transition effectuée sur le système correspond donc à l'ensemble des transitions effectuées sur les composants qui ont évolué. Dans cette définition, nous vérifions que :

- il n'y a qu'un seul message provenant de l'extérieur,
- tous les messages reçus ont été envoyés sur le point de connexion connecté au point sur lequel le message a été reçu, et que ce message est bien un message de |,
- tous les messages envoyés sur des points de connexion connectés ont été reçus par les composants sur les points connectés aux premiers, et que les messages reçus sont bien des messages de |.

7.2.4.3 Comportement explicite du système

Le modèle décentralisé du système définit complètement le comportement du système. Ce comportement est implicite, mais il est possible de calculer le comportement explicite du modèle.

Tout d'abord, il est nécessaire d'introduire la notion d'ε-transition. Une ε-transition correspond au fait qu'un composant ne reçoit pas de message (et donc n'envoie aucun message ni ne change d'état).

Définition 7.8 (Produit libre)**.**
Soit $\Sigma_i = \langle Q_i, E_i, P_i, T_i, Q_i^o \rangle$ *les* n *composants* c_i. *Le* produit libre *des* n *automates communicants* Σ_i *est un automate* $\Sigma = \langle Q, E, P, T, Q^o \rangle$ *où :*

- $Q = Q_1 \times \cdots \times Q_n$,

- $E = E_1 \cup \cdots \cup E_n$,
- $P = P_1 \cup \cdots \cup P_n$,
- $T = (T_1 \cup \{\varepsilon\}) \times \cdots \times (T_n \cup \{\varepsilon\})$ *est l'ensemble des transitions* $(q_1, \ldots, q_n) \overset{(m_1,\ldots,m_n)}{\longrightarrow} (q'_1, \ldots, q'_n) = (q_1 \overset{m_1}{\longrightarrow} q'_1, \ldots q_n \overset{m_n}{\longrightarrow} q'_n)$, *où* $q_i \overset{m_i}{\longrightarrow} q'_i$ *est une transition de* T_i *ou une* ε*-transition,*
- $Q^o = Q^o_1 \times \cdots \times Q^o_n$.

Le produit libre correspond au comportement du système sans contrainte de synchronisation sur les connexions.

Soit t une transition $(t_1, \ldots, t_n)$, avec $t_i = (q_i, (p_i, e_i), \{((p_{i,1}, e_{i,1}), \ldots, (p_{i,k}, e_{i,k}))\}, q'_i)$ ou $t_i = \varepsilon$. On note $t = (q, eed, eeg, q')$ avec $q = (q_1, \ldots, q_n)$, $q' = (q'_1, \ldots, q'_n)$, eed est l'ensemble des messages reçus par les composants et eeg est l'ensemble des messages envoyés par les composants. On peut calculer ces ensembles *via* ces formules : $eed = \bigcup_i \{(c_i, p_i, e_i)\}$ et $eeg = \bigcup_{i,j} \{(c_i, p_{i,j}, e_{i,j})\}$ ($\forall i$ tel que t_i n'est pas une ε-transition).

Définition 7.9 (Synchronisation sur une connexion)**.**
Une transition $t = (q, eed, eeg, q')$ *est* synchronisée *sur une connexion* $co = ((c_j, p_j), (c_k, p_k), |)$ *si :*

- $(\exists e_j, (c_j, p_j, e_j) \in eeg) \Rightarrow (\exists e_k, (c_k, p_k, e_k) \in eed \wedge (e_j, e_k) \in |)$,
- $(\exists e_k, (c_k, p_k, e_k) \in eed) \Rightarrow (\exists e_j, (c_j, p_j, e_j) \in eeg \wedge (e_j, e_k) \in |)$.

La synchronisation sur une connexion vérifie qu'un message envoyé sur l'un des points de la connexion a bien été reçu sur l'autre point de la connexion, et réciproquement.

Définition 7.10 (Synchronisation sur une topologie)**.**
Une transition $t = (q, eed, eeg, q')$ *est synchronisée sur une topologie Top si :*

- *elle est synchronisée sur chaque connexion de la topologie,*
- $\forall (c, p, e),\ (c, p, e) \in eed \Rightarrow (\exists c_1, p_1, c_2, |,\ ((c_1, p_1), (c_2, p), |) \in Top) \vee p = p_i^{ext}$,
- $\exists! c,\ \exists e,\ (c, p^{\text{ext}}, e) \in eed$.

La première proposition de la définition 7.10 s'assure que les messages sont synchronisés sur les connexions. La deuxième proposition vérifie que les messages reçues l'ont été *via* une connexion (ou un point externe). Remarquons qu'un message ne peut être reçu sur un point de connexion qui n'est pas connecté, tandis qu'un message peut être envoyé sur un point de connexion qui n'est pas connecté (dans ce cas, le message est perdu). Remarquons aussi qu'un message ne peut être envoyé sur un point de connexion connecté si le composant devant recevoir ce message n'a pas de transition étiquetée par la réception de ce message. Enfin, la dernière proposition indique qu'il est nécessaire qu'il y ait un événement externe (et un seul).

Il est à présent possible de donner le modèle explicite du système.

Définition 7.11 (Comportement explicite du système)**.**
Le comportement explicite du système $(Comp, LMod, Top)$ *est l'automate* $\Sigma' = \langle Q, E, P, T', Q^o \rangle$ *du produit libre* $\Sigma = \langle Q, E, P, T, Q^o \rangle$ *tel que* $T' \subseteq T$ *est l'ensemble des transitions synchronisées sur la topologie Top.*

Exemple Considérons le cas, dans notre exemple, où notre système comprend une pompe et un tuyau connectés. Le modèle du système est alors le suivant : $(Comp, LMod, Top)$ avec :

- $Comp = \{P, T1\}$,
- $LMod$ associe à P le modèle présenté à la figure 7.4 et à $T1$ le modèle présenté à la figure 7.3,
- $Top = \{((P, out), (T1, in), Id)\}$ (la connexion présentée à la page 139).

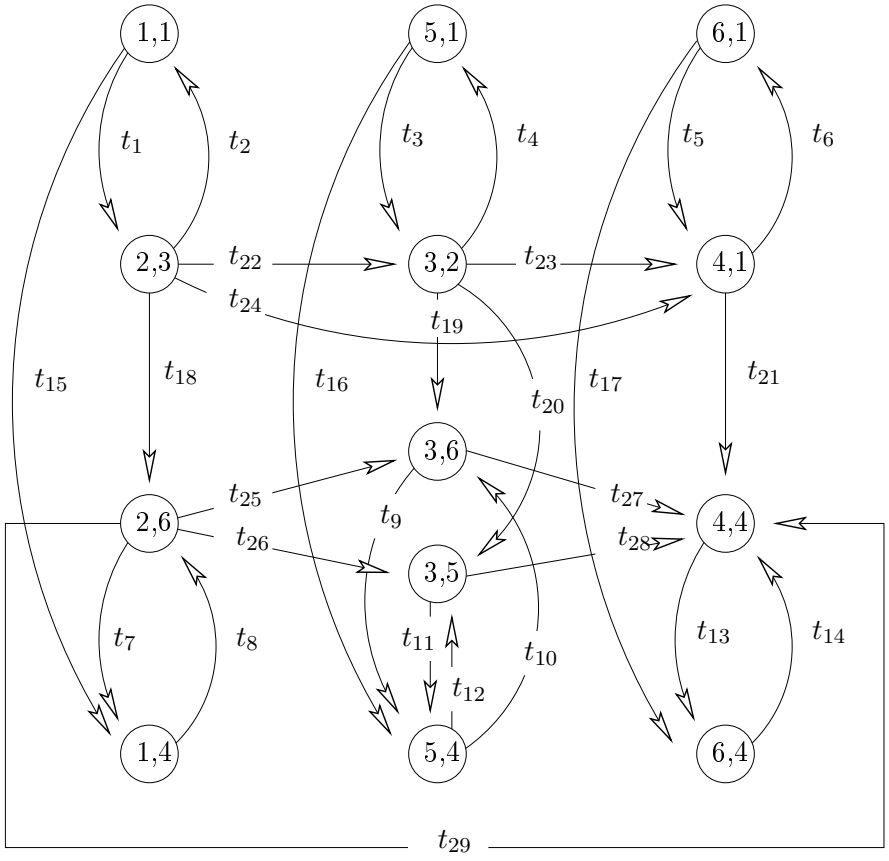

FIG. 7.6 – Modèle explicite du système

Le modèle du système peut être calculé de manière explicite. Le modèle obtenu est présenté à la figure 7.6 (les transitions sont détaillées dans la table 7.2). La taille du modèle est exponentielle avec le nombre de composants. Il est donc évident que le modèle explicite ne doit être en pratique jamais calculé.

7.2.5 Équivalence entre les deux modélisations

On peut transformer le modèle du composant $LMod(c)$ par un automate $Mod_i = (Q, E, T, I, F)$ comme utilisé dans les précédents chapitres. L'ensemble E des événe-

Transition	Étiquette
t_1	*P:ext:on/{T1:out:h}*
t_2	*P:ext:off/{T1:out:z}*
t_3	*P:ext:on/{T1:out:l}*
t_4	*P:ext:off/{T1:out:z}*
t_5	*P:ext:on/{}*
t_6	*P:ext:off/{}*
t_7	*P:ext:off/{T1:out:z}*
t_8	*P:ext:on/{T1:out:l}*
t_9	*P:ext:off/{T1:out:z}*
t_{10}	*P:ext:on/{T1:out:l}*
t_{11}	*P:ext:off/{}*
t_{12}	*P:ext:on/{}*
t_{13}	*P:ext:off/{}*
t_{14}	*P:ext:on/{}*
t_{15}	*T1:ext:f/{}*
t_{16}	*T1:ext:f/{}*
t_{17}	*T1:ext:f/{}*
t_{18}	*T1:ext:f/{T1:out:l}*
t_{19}	*T1:ext:f/{T1:out:l}*
t_{20}	*T1:ext:f/{T1:out:z}*
t_{21}	*T1:ext:f/{}*
t_{22}	*P:ext:f1/{T1:out:l}*
t_{23}	*P:ext:f2/{T1:out:z}*
t_{24}	*P:ext:f2/{T1:out:z}*
t_{25}	*P:ext:f1/{T1:out:l}*
t_{26}	*P:ext:f1/{T1:out:z}*
t_{27}	*P:ext:f2/{T1:out:z}*
t_{28}	*P:ext:f2/{}*
t_{29}	*P:ext:f2/{T1:out:z}*
t_{30}	*P:ext:f1/{T1:out:l}*
t_{31}	*P:ext:f1/{T1:out:z}*

TAB. 7.2 – Table des étiquettes de transition de la figure 7.6

ments est défini comme suit :

- pour tout port p du modèle $LMod(c)$ non connecté, pour tout événement $e \in E$, on a $(c, p, e) \in E$,
- pour tout port p du modèle connecté au composant c' par le port p' avec l'ensemble de synchronisation $| \; (((c, p), (c', p'), |) \in Top)$, pour tout événement $e \in E$, pour tout événement e' de $LMod(c')$, on a $\{(c, p, e), (c', p', e')\} \in E$.

Un événement est donc défini comme un événement sur un composant ou un ensemble de deux événements sur deux composants. Par ailleurs, nous notons $F = Q$, et I est

l'ensemble des états atteignables par la reconfiguration. $T = \{(q, l, q')\}$ est tel qu'il existe une transition $(q, (p, e), \{(p_k, e_k)\}, q')$ et l est l'ensemble d'événements comprenant (c, p, e) et pour chaque k un événement produit à l'aide de p_k et e_k et synchronisé vis à vis de $|$. Remarquons qu'il se peut qu'aucun événement ne puisse être produit à l'aide de p_k et e_k. Le modèle global *Mod* du système peut être alors obtenu en synchronisant les modèles locaux Mod_i en s'assurant que chaque transition de l'automate obtenu ne comporte qu'un seul événement sur un port externe (puisqu'un seul événement exogène peut avoir lieu à un moment donné).

Considérons l'exemple du tuyau présenté à la figure 7.3 et à la table 7.1). L'ensemble d'événements E du modèle simplifié est : *{(T1,ext,h),(T1,ext,l),(T1,ext,z), {(P,out,h),(T1,in,h)}, {(P,out,l),(T1,in,l)}, {(P,out,z),(T1,in,z)}, (T1,out,h), (T1,out,l), (T1,out,z)}*. Considérons maintenant la transition t_1 = (3,*in:z*/{*out:z*},1) du modèle du tuyau. Le message *out:z* émis n'est reçu par aucun composant. Ainsi, l'émission du message est représenté par l'événement e_1 = *(T1,out,z)* $\in E$. Le message *in:z* a été émis par le composant P sur le point de connexion *out*. Compte-tenu que l'ensemble de synchronisation sur la connexion est l'identité, le message émis par P ayant généré la réception de z est forcément z. Ainsi, un seul événement représente la réception du message par le tuyau *T1* : e_2 = *{(P,out,z),(T1,in,z)}*. La transition t_1 est donc représentée par la transition $(3, l, 1)$ où $l = \{e_1, e_2\}$.

La table 7.3 donne les transitions du modèle du tuyau dans la modélisation simplifiée. Ainsi, l'étiquette de la transition t_1 (*{{(P,out,z),(T1,in,z)},(T1,out,z)}*) peut se lire de la manière suivante : l'émission du message z sur le port *out* de P correspond à la réception instantanée du message z sur le port *in* de *T1* et provoque l'envoi du message z sur le port *in* de *T1*.

7.3 Extension de la modélisation pour la reconfiguration

La modélisation présentée en section 7.2 permet de considérer la reconfiguration d'un système mais n'est pas claire sur les conséquences de cette reconfiguration. Comme nous l'avons dit lors du chapitre 6 consacré à la reconfiguration, une reconfiguration change l'état du système et ne peut être effectuée que dans certaines conditions.

Nous donnons ici d'abord la définition d'une reconfiguration pour les systèmes modélisés comme dans 7.2. Puis nous donnons nos hypothèses sur les reconfigurations.

Dans notre contexte, les modifications *atomiques* qui peuvent avoir lieu sur le système sont les suivantes :

- ajout d'un composant (non connecté),
- suppression d'un composant (non connecté),
- ajout d'une connexion,
- suppression d'une connexion.

Les autres modifications du système peuvent se décrire par ces modifications atomiques. Par exemple, le remplacement d'un composant obsolète par un nouveau composant peut se décrire de la manière suivante : suppression des connexions menant au composant ob-

Transition	Étiquette
t_1	*{{(P,out,z),(T1,in,z)},(T1,out,z)}*
t_2	*{{(P,out,h),(T1,in,h)},(T1,out,h)}*
t_3	*{{(P,out,z),(T1,in,z)},(T1,out,z)}*
t_4	*{{(P,out,l),(T1,in,l)},(T1,out,l)}*
t_5	*{{(P,out,h),(T1,in,h)},(T1,out,h)}*
t_6	*{{(P,out,l),(T1,in,l)},(T1,out,l)}*
t_7	*{{(P,out,l),(T1,in,l)},(T1,out,l)}*
t_8	*{(T1,ext,f)}*
t_9	*{(T1,ext,f),(T1,out,z)}*
t_{10}	*{(T1,ext,f),(T1,out,l)}*
t_{11}	*{(T1,ext,f),(T1,out,l)}*
t_{12}	*{{(P,out,l),(T1,in,l)}}*
t_{13}	*{{(P,out,l),(T1,in,l)},(T1,out,l)}* et *{{(P,out,h),(T1,in,h)},(T1,out,l)}*
t_{14}	*{{(P,out,z),(T1,in,z)}}*
t_{15}	*{{(P,out,l),(T1,in,l)},(T1,out,l)}* et *{{(P,out,h),(T1,in,h)},(T1,out,l)}*
t_{16}	*{{(P,out,z),(T1,in,z)}}*
t_{17}	*{{(P,out,f),(T1,in,z)}}*
t_{18}	*{{(P,out,l),(T1,in,l)},(T1,out,l)}* et *{{(P,out,h),(T1,in,h)},(T1,out,l)}*
t_{19}	*{{(P,out,f),(T1,in,z)}}*

TAB. 7.3 – Table des étiquettes de transition pour la modélisation simplifiée de la figure 7.3

solète, suppresion du composant obsolète, ajout du nouveau composant, rétablissement des connexions vers le nouveau composant.

On considère que lorsqu'un nouveau type de composant est créé, son modèle existe et il n'est pas nécessaire d'indiquer qu'un nouveau modèle est créé dans la reconfiguration.

Définition 7.12 (Reconfiguration)**.**
Soit $(Comp, LMod, Top)$*, le modèle d'un système* $\mathcal{S}$*. Une reconfiguration* $\mathcal{R}$ *du système* $\mathcal{S}$ *est un n-uplet* (AS, AA, AD, AC) *où :*

- AS *et* AA *sont des ensembles de paires (composant,modèle), la liste des composants étant obtenue par* $C(AS)$ *et* $C(AA)$ *(*$C(AS) \cap C(AA) = \emptyset$*), et*
- AD *et* AC *sont des ensembles de connexions (*$AD \subseteq Top$*),*

et tel que $(Comp', LMod', Top')$ *avec :*

- $Comp' = Comp \setminus C(AS) \cup C(AA)$,
- $LMod' = LMod \setminus AS \cup AA$,
- $Top' = Top \setminus AD \cup AC$,

est un modèle de système.

AS est l'ensemble des composants à supprimer (*actions de suppression*) ; AA est l'ensemble des composants à ajouter (*actions d'ajout*) ; AD est l'ensemble des connexions à supprimer (*actions de déconnexion*) ; et AC est l'ensemble des connexions à ajouter

(*actions de connexion*). On note $\mathcal{R}((Comp, LMod, Top)) = (Comp', LMod', Top')$ avec les valeurs données dans la définition 7.12.

Soit $\mathcal{R}$, une reconfiguration, on note $AS_{\mathcal{R}}$ l'ensemble des composants à supprimer de la reconfiguration $\mathcal{R}$, *etc.*

Nous avons défini le modèle du système à l'issue de la reconfiguration, mais il est nécessaire de déterminer le nouvel état du système.

Nous considérons qu'une reconfiguration ne peut pas être exécutée quand le système est dans un état quelconque. Par exemple, il n'est clairement pas *sain* de déconnecter un tuyau qui est en train de délivrer un flux de liquide. Il est raisonnable de penser qu'un système de protection (ou le comportement de l'opérateur) empêche une telle action. Plus précisément, le fait qu'une reconfiguration soit saine dépend de ses points de connexion, et de l'état des composants à qui appartiennent ces points de connexion.

C'est pourquoi, avant de définir cette notion, nous étendons le modèle comportemental en associant à chaque point de connexion un ensemble d'états dans lesquels une connexion ou déconnexion du point de connexion est autorisée.

Le modèle du composant est étendu par l'addition de G dans la définition qui suit :

Définition 7.13 (Modèle étendu d'un composant)**.**
Le modèle d'un composant est décrit par un automate communicant $\Sigma = \langle Q, E, P, T, G, q^o \rangle$ *où :*

- Q *est l'ensemble des états du composant,*
- E *est l'ensemble des messages qui peuvent être reçus ou émis par le composant,*
- P *est l'ensemble des points de connexion du composant,* p^{ext} *est le point externe,*
- $T \subset (Q \times (P \times E) \times (P \times E)^* \times Q$ *est l'ensemble des transitions,*
- $G \in P \setminus \{p^{\text{ext}}\} \rightarrow 2^Q$ *est une fonction qui associe à chaque point de connexion un ensemble d'états qui autorisent une connexion ou une déconnexion sur ce point,*
- q^o *est l'état initial.*

Dans notre exemple, l'ensemble des états dans lesquels les connexions de la pompe peuvent être reconfigurées sont données par $G_P(out) = \{1, 4, 5, 6\}$ (*out* est l'unique point de connexion interne du modèle de la pompe). Les états 1, 4 et 6 de la pompe indiquent que la pompe est éteinte, tandis que l'état 5 est un état où la pompe est en panne (et ne délivre ainsi aucun flot). On pourrait également considérer que l'état 5 n'est pas sain pour le point de connexion *out*. Le tuyau dispose de deux points de connexion interne : *in* et *out*. On a : $G_T(in) = \{1, 4\}$ et $G_T = \{1, 4, 5\}$, les états 1 et 4 correspondant à des états où aucun flot n'entre dans le tuyau, et l'état 5 l'état où aucun flot ne sort du tuyau alors qu'un faible flot entre.

Une reconfiguration *saine* est définie comme suit :

Définition 7.14 (Reconfiguration saine)**.**
Une reconfiguration R *est* saine *dans l'état* q *si* $\forall((c_1, p_1), (c_2, p_2), |) \in AD_{\mathcal{R}} \cup AC_{\mathcal{R}}$, $\forall i \in \{1, 2\}$, *l'état du composant* c_i *est compris dans* $G_{c_i}(p_i)$ *:* $q(c_i) \in G_{c_i}(p_i)$.

La sanité d'une reconfiguration indique que si on effectue une connexion ou une déconnexion sur un point de connexion, alors l'état du système doit être dans un état sain pour ce point de connexion.

La définition de sanité est suffisante pour considérer que l'état d'un composant n'est pas modifié par une reconfiguration saine.

Lemme 7.1.
L'état d'un composant n'est pas modifié par une reconfiguration saine.

Dans notre exemple, nous voyons que les composants sont connectés lorsqu'il ne transmettent ni ne reçoivent aucun flux d'entrée. On voit donc que leur état n'a pas de raison d'être modifié par une connexion ou un déconnexion. Remarquons aussi qu'une reconfiguration ne peut pas provoquer un endommagement de la pompe ou du tuyau.

Le nouvel état du système n'est en revanche pas défini pour des reconfigurations non saines.

Il est possible de modifier le modèle du système pour considérer que seules des reconfigurations saines ont lieu. Par exemple, si une reconfiguration dans un état q est censée générer un comportement c juste après la reconfiguration (par exemple, la reconnaissance d'un nouveau périphérique par un ordinateur) on pourra représenter le comportement du composant comme présenté à la figure 7.7. Avant la reconfiguration, on atteint l'état q_2 par l'occurrence d'un événement fictif (on peut interpréter cet événement comme une annonce de la reconfiguration). Seul l'état q_2 autorise la reconfiguration $q_2 \in G(p)$). Puis, on quitte l'état q_2 par une transition (ou une séquence de transitions) étiquetée(s) par le comportement c. On voit ainsi que pour faire évoluer le système après avoir une reconfiguration, il est nécessaire d'effectuer c. On peut également ajouter des observations fictives sur les transitions supplémentaires (et rajouter ces observations fictives à l'ensemble des observations quand on sait qu'une reconfiguration a été effectuée) pour s'assurer qu'on n'emprunte pas l'état q_2 en dehors des reconfigurations et être certain qu'on ne reste pas dans l'état q_2 à l'issue de la reconfiguration.

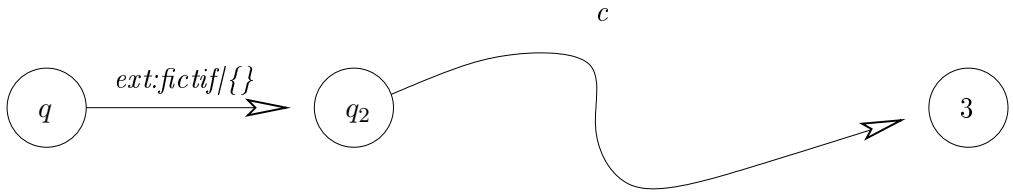

FIG. 7.7 – Modélisation saine d'une reconfiguration non saine

Puisqu'il est possible de modéliser des comportements non sains de manière saine, on considère que le système est modélisé de manière à permettre l'hypothèse suivante :

Hypothèse 7.7.
Toutes les reconfigurations sont saines.

Cette hypothèse permet de connaître l'ensemble des états q' du système à l'issue d'une reconfiguration $\mathcal{R}$ à partir de l'état q du système avant la reconfiguration. Soit $(Comp, LMod, Top)$ le modèle du système avant la reconfiguration et $(Comp', LMod', Top')$ le modèle après la reconfiguration. Soit q un état du système avant la reconfiguration. On note $q \overset{\mathcal{R}}{\mapsto} q'$ avec :

- $\forall (c, m) \in AS$, $q'(c)$ n'est pas défini,
- $\forall (c, m) \in AA$, $q'(c) = q_0$ où q_0 est un état initial de m, et
- $\forall c \in Comp \cap Comp'$, $q'(c) = q(c)$.

7.4 Représentation des observations et des reconfigurations

Dans RAG$_e$, nous avons considéré les hypothèses présentées dans 2.2 que nous avons augmentées. Nous n'utilisons pas de manière explicite un automate.

Une observation émise par le système est toujours reçue et certaine. De plus, lorsque le diagnostic est fait à une date t_i, on dispose d'un sur-ensemble $\mathcal{O}(t_i)$ des observations émises avant t_i : c'est-à-dire que pour certaines observations, on ne sait pas si l'observation a été émise avant t_i, mais on sait que toute observation émise avant t_i est présente dans $O(t_i)$ ($o \prec t_i \Rightarrow o \in \mathcal{O}(t_i)$). On dispose également de $P(t_i)$ l'ensemble des observations qui ont été émises sûrement avant t_i ($o \in P(t_i) \Rightarrow o \prec t_i$).

Les observations sont ordonnées par composant. Lorsqu'on reçoit une observation o_1 puis une observation o_2 émises par un même composant, alors on sait que les observations ont été émises dans ce même ordre. En revanche, on ne dispose pas d'informations sur l'ordre d'observations o_1 et o_2 de deux composants différents (excepté si $\exists t_i$, $o_1 \prec t_i$ et $t_1 \prec o_2$).

Les observations sont fournies dans des fichiers comme celui présenté sur la figure 7.8. Dans ce fichier, la première ligne indique qu'il s'agit d'une fenêtre d'observations. Les lignes suivantes donnent les observations reçues pour chaque composant. Ainsi, le superviseur a-t-il observé le message *a* puis le message *a1* sur le composant *c1*, et l'observation *b* sur le composant *c2*. Le message *a* a été émis avant *a1*, mais l'ordre entre *a* ou *a1* et *b* est inconnu.

Après le premier `END`, sont indiqués le nombre d'observations qui ont certainement été émises avant la fin de cette fenêtre, ici un pour *c1* et zéro pour *c2*. Ce nombre d'observations se cumule avec le nombre obtenu à la précédente fenêtre. Considérons par exemple qu'à l'issue de la précédente fenêtre, 17 observations aient été reçues du composant *c1* et que les 15 premières ont été certainement émises avant la fenêtre courante. Alors, à l'issue de la fenêtre courante, 19 observations ont été reçues (puisqu'on ajoute les événements *a* et *a1*) et *16* ont été certainement émises.

Les reconfigurations sont présentées dans un fichier dont un exemple est donné à la figure 7.9. Dans ce fichier, la première ligne indique que le fichier décrit une reconfiguration. Les premières lignes sont utilisées pour ajouter deux nouveaux modèles à la base de modèles. Les deux lignes suivantes instancient deux nouveaux composants : le composant *c1* modélisé par *m1* et le composant *c2* modélisé par *m2*. La ligne suivante crée une connexion appelée *connexion2* entre le point de connexion *out* de *c1* et le point de

```
OBSERVATIONS
c1 a ;
c1 a1 ;
c2 b ;
END ;
c1 1
c2 0
END ;
```

FIG. 7.8 – Exemple de fichier d'observations

connexion *in* du composant *c2*. Remarquons que cette connexion est considérée comme synchronisée sur l'ensemble *identité*. Une extension de la notation du format des fichiers est prévue pour permettre de manipuler des synchronisations plus complexes. La ligne `DELETE PI1` supprime le composant *PI1* et la ligne suivante supprime la connexion nommée *connexion1*.

```
RECONFIGURATION
LOAD data/models/c1.rage.xml
LOAD data/models/c2.rage.xml
CREATE m1 c1
CREATE m2 c2
CONN connexion2 c1 out c2 in
DELETE PI1
DECONN connexion1
END ;
```

FIG. 7.9 – Exemple de fichier de reconfiguration

7.5 Algorithmes

Nous voyons à présent les algorithmes utilisés dans RAG_e. Nous présentons tout d'abord l'algorithme de calcul du diagnostic local et ensuite l'algorithme de fusion des diagnostics. Ensuite, nous voyons comment est calculé le diagnostic d'une reconfiguration.

Un diagnostic de sous-système (qu'il s'agisse d'un composant, d'un sous-système ou du système) est un automate $\Delta = (Q, T, I, F)$ où Q est l'ensemble des états, T l'ensemble des transitions, I l'ensemble des états initiaux et F l'ensemble des états finaux. Une transition se note $t = (q, ex, eer, eei, eee, eo, q')$ où :

- ex est l'événement exogène déclenchant la transition s'il fait partie du sous-système, ex vaut *null* sinon ;

- *eer* est l'ensemble des événements reçus sur des ports par des connexions internes au système mais externes au sous-système ;
- *eei* est l'ensemble des événements internes au sous-système (chaque événement comporte alors un événement d'émission et un événement de réception) ;
- *eee* est l'ensemble des événements émis par le sous-système sur des ports qui ne sont pas connectés de manière interne au sous-système ;
- *eo* est l'ensemble des observations émises par la transition.

Les ports et l'ensemble des événements sont sous-entendus dans la suite.

7.5.1 Diagnostic local

La principale différence par rapport aux algorithmes proposés dans [Pen02] est donc la prise en compte de fenêtres non sûres. Nous considérons que l'on connaît un sur-ensemble des observations émises pendant la fenêtre temporelle, les observations supplémentaires provenant soit de la fenêtre précédente, soit de la fenêtre suivante.

Dans un diagnostic local, l'état de l'automate des observations est l'ensemble des observations expliquées. On note $\forall k$, $P(t_k)$ l'ensemble des observations sûrement émises avant t_k, et $\forall k$, $O(t_k)$ l'ensemble des observations qui ont pu être émises avant t_k. Ces ensembles étant ordonnés, on peut également les considérer comme des séquences. Rappelons que $\forall k$, $P(t_k) \subseteq O(t_k)$, $P(t_{k-1}) \subseteq P(t_k)$ et $O(t_{k-1}) \subseteq O(t_k)$. Ces ensembles peuvent être considérées comme des séquences puisque les observations sont complètement ordonnées pour un composant donné.

Algorithme 3 Diagnostic local

Fonction **diagnostic_local**$(P(t_{i-1}), O(t_{i-1}), P(t_i), O(t_i), F^{i-1})$
$ec = \emptyset$
$O = P(t_{i-1})$
$ec = ec \cup \{q \mid q \in F^{i-1} \wedge (\exists q',\ q = (q', O))\}$
$ec =$ **développe_invisibles**(ec)
Tant que $O \neq O(t_i)$ **faire**
  $o =$ observation suivante dans la séquence O dans $O(t_i)$
  $ec =$ **développe_observation**(ec, o)
  $O = O \cup \{o\}$
  $ec = ec \cup \{q \mid q \in F^{i-1} \wedge (\exists q',\ q = (q', O))\}$
  $ec =$ **développe_invisibles**(ec)
Fin tant que
$I^i = F^{i-1}$
$F^i = \{q \mid q \in Q^i \wedge (\exists q', O,\ q = (q', O) \wedge P(t_i) \subseteq O \subseteq O(t_i))\}$
simplifie(Δ)
return Δ

Le calcul du diagnostic local $\Delta_i^j = (Mod_i \otimes Obs^j)[F^{i-1}]$ où F^{i-1} est la projection de l'ensemble des états finaux de la précédente fenêtre sur Δ_i^{j-1} est présenté à l'algorithme 3. L'algorithme fonctionne ainsi. On part de l'ensemble $O = P(t_{i-1})$ qui

est l'ensemble minimal d'observations reçues avant la fenêtre. Ainsi, il est impossible de trouver un état comportant un ensemble plus faible. L'ensemble des états courants (noté ec) du diagnostic est l'ensemble F^{i-1} des états finaux de la précédente fenêtre qui ont un ensemble égal à $P(t_{i-1})$. On calcule les états accessibles par des transitions *invisibles* suivies d'une transition comportant l'observation suivante. Au tour de boucle suivant, et tant que l'ensemble des observations O est inclus dans $O(t_{i-1})$, on ajoute à l'ensemble ec d'états courants les états de F^{i-1} comportant la séquence O. Remarquons que dans notre cadre, ce n'est pas indispensable. En effet, s'il existe un état de F^{i-1} qui correspond à l'ensemble des observations $O \supset P(t_{i-1})$, alors cet état était accessible depuis au moins un état correspondant à l'ensemble des observations de $P(t_{i-1})$ dans le diagnostic local de la précédente fenêtre. Or, puisque le modèle du composant n'a pas changé, cet état est également accessible dans le diagnostic local. Ainsi, à chaque tour de boucle, l'opération $ec = ec \cup \{q \mid q \in F^{i-1} \wedge (\exists q',\ q = (q', O))\}$ ne modifie pas la valeur de ec.

On peut remarquer que la synchronisation des observations et du modèle est faite en même temps que la restriction de l'ensemble des états initiaux. Cette restriction est faite en ne considérant pour états initiaux que les états de F^{i-1}.

Algorithme 4 Développement des transitions invisibles

Fonction **développe_invisibles**(ec)
$ouvert = ec$
$fermé = \emptyset$
Tant que ($ouvert \neq \emptyset$) **faire**
  $(q, O) =$ **retirer_élement**(ouvert)
  $fermé = fermé \cup \{(q, O)\}$
  Pour chaque $t = (q, ed, eeg, q') \in T^{Mod_i}$ tel que eeg ne comporte pas d'événement sur le port p^{obs} **faire**
    $T = T \cup \{$**construit_transition_locale**$(t, O, O)\}$
    Si $(q', O) \notin \{ouvert \cup fermé\}$ **alors**
      $Q = Q \cup \{(q', O)\}$
      $ouvert = ouvert \cup \{(q', O)\}$
    Fin si
  Fin pour
Fin tant que
return $fermé$

La fonction développant les transitions invisibles est présentée dans l'algorithme 4. Il s'agit d'un simple algorithme de recherche *ouvert-fermé*. Le développement de l'observation o est présenté algorithme 5. Les deux algorithmes font référence à la fonction **construit_transition_locale**(t, O, O'). Cette fonction génère la transition entre l'état (q, O) et (q', O') de l'automate de diagnostic en calculant les valeurs de ex, eer, eei, eee et eo.

La simplification se fait simplement en vérifiant par un algorithme arrière quels états permettent d'atteindre un état final.

Algorithme 5 Développement des transitions représentant l'observation o

Fonction **développe_observation**(ec, o)
$result = \emptyset$
Pour chaque $(q, O) \in ec$ **faire**
  Pour chaque $t = (q, l, q') \in T^{Mod_i}$ tel que l génère l'observation o **faire**
    $T = T \cup \{$**construit_transition_locale**$(t, O, O \cup \{o\})\}$
    $Q = Q \cup \{(q', O \cup \{o\})\}$
    $result = result \cup \{(q', O \cup \{o\})\}$
  Fin pour
Fin pour
return *result*

7.5.2 Fusion des diagnostics

Soit Δ_1 le diagnostic d'un sous-système et Δ_2 le diagnostic d'un autre sous-système tel que *Top* est l'ensemble des connexions entre les deux sous-systèmes. L'algorithme de synchronisation des diagnostics se base sur l'opération de synchronisation de deux transitions présenté à l'algorithme 6 (page 155). Tout d'abord, la synchronisation n'est possible que si un seul ou aucun événement exogène n'apparait sur les deux transitions. On vérifie que tous les messages reçus sur les connexions *Top* ont bien été émis, et on construit les événements internes asociés (en supprimant les deux événements reçu et émis). Par la suite, s'il y a des événements émis sur une connexion de *Top* mais non reçus, alors la synchronisation est impossible.

La synchronisation des diagnostics de deux sous-systèmes est l'algorithme 7. L'ensemble des ouverts est initialisé à $(I_1 \times I_2) \cap F^{i-1}$ ce qui permet d'effectuer la restriction. L'algorithme suit une exploration *ouvert-fermé* classique en utilisant pour chaque transition une transition de T_1 et une transition de T_2. Rappelons qu'il est possible qu'une seule transition soit empruntée, ce qui explique l'introduction des transitions vides $(q, null, \emptyset, \emptyset, \emptyset, \emptyset, q)$.

7.5.3 Reconfiguration

L'algorithme de reconfiguration est extrêmement simple. En plus de mise à jour du modèle, on vérifie pour chaque état final du diagnostic de la précédente fenêtre que la reconfiguration est possible. Pour chaque état, on calcule l'état atteint après la reconfiguration, et cet état est ajouté à l'ensemble des états initiaux de la fenêtre suivante.

7.6 Conclusion

Dans ce chapitre, nous avons présenté l'outil de diagnostic RAG$_e$ que nous avons développé utilisant les résultats de cette thèse. Nous avons donné un formalisme plus complexe pour la modélisation des automates et montré la correspondance entre ce formalisme et le précédent. Nous avons montré comment il était possible de considérer

Algorithme 6 Synchronisation de deux transitions

Fonction **synchronise**(t_1, t_2, Top)
$t_1 = (q_1, ex_1, eer_1, eei_1, eee_1, eo_1, q'_1)$
$t_2 = (q_2, ex_2, eer_2, eei_2, eee_2, eo_2, q'_2)$
Si $(ex_1 \neq null) \wedge (ex_2 \neq null)$ **alors**
  Synchronisation impossible
Sinon si $ex_1 \neq null$ **alors**
  $ex = ex_1$
Sinon
  $ex = ex_2$
Fin si
$eer = eer_1 \cup eer_2$
$eei = eei_1 \cup eei_2$
$eee = eee_1 \cup eee_2$
$eo = eo_1 \cup eo_2$
Pour chaque $e \in eer$ tel qu'il existe une connexion $co \in Top$ qui correspond au port de e **faire**
  Si $\exists e' \in eei$ qui se synchronise sur co avec e **alors**
    $eei = eei \cup \{(e, e')\}$
    $eer = eer - e$
    $eee = eee - e'$
  Sinon
    Synchronisation impossible
  Fin si
Fin pour
Si $\exists e' \in eee$ tel qu'il existe une connexion $co \in Top$ qui correspond au port de e' **alors**
  Synchronisation impossible
Fin si
Si $ex \cup eer = \emptyset$ **alors**
  Synchronisation impossible
Fin si
return $((q_1, q_2), ex, eer, eei, eee, eo, (q'_1, q'_2))$

Algorithme 7 Synchronisation de deux diagnostics de sous-systèmes

Fonction **synchronise**$(\Delta_1, \Delta_2, F^{i-1}, Top)$
$\Delta_1 = (Q_1, T_1, I_1, F_1)$
$\Delta_2 = (Q_2, T_2, I_2, F_2)$
$ouvert = (I_1 \times I_2) \cap F^{i-1}$
$fermé = \emptyset$
$Q = ouvert$, $T = \emptyset$, $I = Q$; $F = \emptyset$
Tant que $ouvert \neq \emptyset$ **faire**
  $(q_1, q_2) =$ **retirer_élement**(ouvert)
  Pour chaque $t_1 \in T \cup \{(q, null, \emptyset, \emptyset, \emptyset, \emptyset, q)\}$ **faire**
    Pour chaque $t_1 \in T \cup \{(q, null, \emptyset, \emptyset, \emptyset, \emptyset, q)\}$ **faire**
      Si $t =$ **synchronise**(t_1, t_2) existe et mène à un état $q' = (q'_1, q'_2)$ **alors**
        $T = T \cup t$
        Si $q' \notin ouvert \cup fermé$ **alors**
          $ouvert = ouvert \cup \{q'\}$
          Si $q'_1 \in F_1 \wedge q'_2 \in F_2$ **alors**
            $F = F \cup q'$
          Fin si
        Fin si
      Fin si
    Fin pour
  Fin pour
Fin tant que
$\Delta = (Q, T, I, F)$
simplifie(Δ)
return Δ

la reconfiguration dans ce formalisme. Enfin, nous avons montré la mise en œuvre dans le diagnostic. Le format de fichier utilisé pour le modèle des composants peut être trouvé en annexe B.

Par rapport à l'outil de diagnostic D-Dyp [Pen02] préexistant, notre application permet de prendre en compte des fenêtres non sûres (dans la mesure où toutes les observations émises avant la date t_i du diagnostic ont été reçues avant la date t où est effectué le diagnostic). L'application permet également de prendre en compte les reconfigurations sur le système.

L'application présentée a été expérimentée, mais le manque d'exemple réel ne nous a pas permis de valider complètement les résultats obtenus. Une interface graphique a été développée par F.-X. Payet au cours d'un stage d'été en 2005.

Conclusion

Bilan

L'objectif de cette thèse était de proposer une méthode permettant de diagnostiquer en-ligne et de manière décentralisée des systèmes à événements discrets reconfigurables avec observabilité incertaine. Nous avons montré de nombreuses approches qui ont été développées dans la littérature pour le diagnostic de systèmes à événements discrets, c'est-à-dire des systèmes qui peuvent être modélisés par des ensembles d'états et d'événements discrets. Le diagnostic doit fournir le comportement du système global pour un superviseur unique du système. Dans cette thèse, nous considérons le système modélisé par un automate noté *Mod*.

L'une des difficultés du diagnostic vient du fait que les observations connues par le superviseur ne sont pas les mêmes que les observations émises par le système. Ceci est dû aux *canaux de communication* qui transmettent les observations de manière imparfaite et aux capteurs qui peuvent générer des erreurs. Ainsi, étant donnée une séquence d'observations reçues, il n'existe pas une seule séquence possible d'observations émises. L'originalité de notre approche est de représenter les observations émises par un automate noté *Obs*. Toute trajectoire sur l'automate des observations est une des séquences possibles d'observations émises. Nous avons montré pour différentes hypothèses comment construire l'automate des observations. Le diagnostic se définit alors comme la synchronisation de l'automate modélisant le comportement du système et de l'automate des observations : $\Delta = Mod \otimes Obs$.

Nous souhaitions pouvoir calculer le diagnostic en-ligne, c'est-à-dire pendant que le système fonctionne. Nous avons montré que pour cela il était nécessaire d'être capable d'effectuer un calcul incrémental du diagnostic (qui peut être également utilisé dans un cadre hors-ligne). Pour permettre ce calcul incrémental, nous avons développé la notion de *chaîne d'automates*. Une chaîne d'automates est une représentation *par morceaux* d'un automate. On peut *découper* un automate en une chaîne et reconstruire l'automate à partir de la chaîne.

Nous avons utilisé ce formalisme et découpé l'automate des observations en une chaîne dont chaque automate correspond au comportement observé pendant une *fenêtre temporelle*. Un raisonnement (synchronisation) peut être effectué pour chaque fenêtre. Il est alors possible d'ajouter de plus en plus de morceaux à la chaîne et d'effectuer

ainsi un calcul incrémental, notamment dans le cadre en-ligne. Nous avons proposé plusieurs types de synchronisations : par morceaux, incrémentale, raffinée, k-raffinée, et présenté les avantages et inconvénients de celles-ci : raisonnement parallèle ou au contraire incrémental, nécessitant plus ou moins de calculs *inutiles* et générant une solution plus ou moins compacte. Nous avons montré l'application de ces résultats pour le diagnostic en-ligne.

En dehors de l'incrémentalité, l'utilisation d'une chaîne d'automates des observations offre une flexibilité plus importante du calcul.

L'approche décentralisée du diagnostic vise à construire un diagnostic de manière locale aux composants du système avant de calculer si nécessaire le diagnostic global du système. Cette approche est beaucoup plus efficace qu'une approche centralisée. Nous avons adapté l'approche décentralisée au diagnostic par chaîne d'automates et montré que l'utilisation des chaînes d'automates pour le diagnostic décentralisé permettait d'augmenter l'efficacité de celui-ci.

Le second objectif important de cette thèse était le diagnostic de systèmes reconfigurables. Nous avons défini le concept de reconfiguration d'un système à diagnostiquer comme la modification de son modèle. Nous avons donné les hypothèses nécessaires au diagnostic de systèmes reconfigurables. Nous avons décidé d'utiliser la flexibilité des chaînes d'automates, où le passage d'une fenêtre à une autre représente l'application d'une reconfiguration. Les espaces d'états du modèle étant différents avant ou après la reconfiguration, le concept de chaîne d'automates a été étendu en *chaîne d'automates hétérogène*.

Enfin, ces résultats ont été validés par l'implémentation d'une plateforme appelée RAG_e qui fournit un diagnostic décentralisé de systèmes reconfigurables par calcul décentralisé.

Perspectives

Les premiers travaux à effectuer concernent la plateforme RAG_e.

Tout d'abord, il conviendra d'utiliser un automate des observations à la place de l'actuelle modélisation. Les ensembles d'observations utilisés dans l'implémentation actuelle doivent donc être remplacés par les états de l'automate des observations. La construction de l'automate devra être effectuée par un module séparé qui génèrera un automate équivalent à la méthode utilisée actuellement. Ce module pourra être étendu pour considérer tous les cas présentés dans le chapitre 2. Pour ce faire, on peut voir la construction de l'automate des observations comme une tâche de diagnostic : il ne s'agit pas de retrouver à partir des observations reçues, les événements de panne mais les observations émises.

Nous avons vu que le diagnostic décentralisé par chaîne d'automates nécessite le calcul de l'ensemble des états finaux de chaque fenêtre. Ce calcul peut être très coûteux.

Cependant, on peut trouver des sous-systèmes dont les états sont indépendants, et avoir une représentation comme suit : $F = F_{ss_1} \times \cdots \times F_{ss_m}$. Ainsi, un calcul intelligent de l'ensemble des états finaux doit permettre de construire et représenter F de manière décentralisée et efficace.

À l'issue de ce travail, il faudra valider complètement la plateforme RAG_e. Pour cela, il faudra considérer une application réelle. En plus de vérifier la justesse des algorithmes proposés, cette expérimentation permettra de vérifier si les algorithmes mis en place sont suffisamment efficaces. En particulier, il conviendra de vérifier si la plateforme permet de diagnostiquer en-ligne, avec un délai raisonnable pour des systèmes d'une certaine taille.

Les résultats présentés en section 5.3 sur l'utilisation conjointe du diagnostic décentralisé et des chaînes d'automates montrent qu'un découpage *intelligent* permet d'obtenir une représentation très compacte des diagnostics. Remarquons que dans l'exemple utilisé, le découpage était déjà effectué. Dans l'approche que nous avons developpée dans cette thèse, le découpage est effectué à certaines dates mais le choix de ces dates n'est pas précisé, excepté pendant les reconfigurations. Un axe de recherche serait de déterminer comment il est possible de trouver les découpages les plus utiles, *a priori* pour accélérer les calculs, ou *a posteriori* pour avoir une représentation compacte. D'autre part, il serait possible d'effectuer un découpage *a posteriori* du diagnostic d'un sous-système et non du système entier, conduisant à des découpages différents d'un sous-système à un autre, comme évoqué dans la conclusion du chapitre 5.

Jusqu'ici, nous considérons que les automates des observations locaux sont tous découpés à la même date. Cependant, tous les capteurs n'ont pas la même dynamique, et certains capteurs peuvent fournir des observations avec un délai très important. Il est très difficile de construire un diagnostic en considérant que certaines observations émises n'ont pas été reçues, puisque la combinatoire est très importante. Aussi, les délais importants sur certains capteurs obligent à raisonner sur tout le système avec ce délai. Une perspective est donc de considérer des dates différentes pour chaque composant. Les fenêtres temporelles sont alors différentes pour chaque composant. La difficulté qui apparaît est de parvenir à fusionner les diagnostics pour ces fenêtres différentes.

Dans cette thèse, nous avons considéré que les reconfigurations sont connues exactement. Notamment, l'ordre de leur exécution est connu parfaitement. Cependant, nous considérons que la reconfiguration est commandé par le superviseur ou par un opérateur centralisé. Dans les systèmes réels, les reconfigurations peuvent être décidées de manière locale, éventuellement sans qu'aucun opérateur n'intervienne. Ainsi, la reconfiguration n'est pas toujours connue du superviseur, ou l'ordre de leurs exécutions n'est pas connu complètement. Il faut alors adapter les algorithmes que nous avons proposés dans cette thèse pour prendre en compte la connaissance partielle de la topologie du système.

La reconfiguration peut être assimilée à un changement de mode de fonctionnement. Lorsque les reconfigurations ne sont pas connues du superviseur, il peut être intéressant de connaître la topologie pour comprendre le fonctionnement interne du système. Par

exemple, dans un réseau de fourniture d'électricité, un composant peut se déconnecter automatiquement lorsqu'il perçoit un dysfonctionnement d'un de ses voisins. Diagnostiquer cette reconfiguration permet de mettre en évidence le dysfonctionnement du composant en panne.

Annexe A

Manipulation d'automates

Cette annexe a pour but de donner les définitions formelles concernant les automates.

A.1 Automate

Il existe de nombreuses notations différentes pour les automates, mais notre choix s'est arrêté sur celle-ci.

Définition A.1 (Automate).
Un automate *est un tuple* (Q, E, T, I, F) *où :*
- Q *est un ensemble fini d'états,*
- E *est un ensemble fini d'événements,*
- $T \subseteq Q \times 2^E \times Q$ *est un ensemble de transitions,*
- $I \subseteq Q$ *est l'ensemble des états initiaux,*
- $F \subseteq Q$ *est l'ensemble des états finaux.*

Une transition se note $t = (q, l, q')$ avec $q \in Q$, $q' \in Q$ et $\emptyset \subset l \subseteq E$. On appelle *étiquette* de l'automate l'ensemble l d'événements. On remarque que cet ensemble est non vide. On considère cependant qu'il existe pour tout $q \in Q$ une transition implicite $(q, \emptyset, q) \in T$. Cette propriété est vraie pour tous les automates que nous construirons même si elle n'est pas indiquée explicitement.

Graphiquement, les états sont représentés par des cercles et les transitions par des flèches. Les états initiaux sont représentés par les flèches ne provenant pas d'un autre état et les états finaux par des doubles cercles. Dans ce rapport, nous représentons généralement les états par un entier ou une lettre en majuscule et les événements par des lettres en minuscule. Cette thèse donne de nombreux exemples d'automates.

A.2 Trajectoire

Les automates permettent de représenter des comportements grâce à des chemins et des trajectoires.

Un chemin représente une séquence de transitions successives sur un automate.

Définition A.2 (Chemin).
Un chemin *sur l'automate* $A = (Q, E, T, I, F)$ *est une double séquence* $((q_0, \ldots, q_m), (l_1, \ldots, l_m))$ *telle que :*
- $\forall i$, $q_i \in Q$ *et*
- $\forall i$, $(q_{i-1}, l_i, q_i) \in T$.

Une trajectoire décrit une évolution depuis un état initial vers un état final.

Définition A.3 (Trajectoire).
Une trajectoire *sur l'automate* $A = (Q, E, T, I, F)$ *est un chemin* $((q_0, \ldots, q_m), (l_1, \ldots, l_m))$ *de* A *tel que* $q_0 \in I$ *et* $q_m \in F$.

Le but d'un automate est de représenter des comportements sous forme de trajectoires. On considère que deux automates sont identiques s'ils ont les mêmes trajectoires. On appelle *automate simplifié* de A, l'automate A dont on a supprimé tous les états et toutes les transitions qui n'apparaissent dans aucune trajectoire. Par la suite, nous considérons que tous les automates calculés (en particulier par synchronisation) sont des automates simplifiés. En effet, les automates simplifiés sont plus petits (puisqu'ils comportent moins d'états et de transitions) et sont donc généralement plus faciles à calculer. Notamment, les algorithmes de synchronisation (voir par la suite) partent généralement de l'ensemble des états initiaux et ne développent que les états accessibles depuis les états initiaux. Cependant, pour des raisons d'implémentation, on pourra éventuellement ne pas calculer un automate simplifié, mais un *sur-automate*. Les propriétés des automates construits énoncées dans ce rapport sont généralement conservées pour des *sur-automates*, mais pas toujours (en particulier pour les chaînes d'automates).

A.3 Synchronisation

La synchronisation d'automates est une opération qui se fait par un ensemble de synchronisation. Cet ensemble indique quels événements doivent apparaitre simultanément sur deux automates à synchroniser. Ici, nous considérons que ces événements sont les événements communs aux deux automates.

Définition A.4 (Synchronisation d'étiquettes).
Soit l_1 *et* l_2 *deux étiquettes de transition. La* synchronisation *de* l_1 *et* l_2 *sur l'ensemble de synchronisation* E, *notée* $\Theta_E(l_1, l_2)$ *et définie si* $l_1 \cap E = l_2 \cap E$, *vaut* $l_1 \cup l_2$.

Deux étiquettes de transition sont synchrones si elles comportent le même sous-ensemble d'événements de E.

Définition A.5 (Synchronisation d'automates).
Soit $A_1 = (Q_1, E_1, T_1, I_1, F_1)$ *et* $A_2 = (Q_2, E_2, T_2, I_2, F_2)$ *deux automates. On appelle* synchronisation *de* A_1 *et* A_2, *noté* $A_1 \otimes A_2$ *l'automate* $A = (Q, E, T, I, F)$ *défini par :*
- $Q = Q_1 \times Q_2$,
- $E = E_1 \cup E_2$,

- $T = \{((q_1, q_2), l, (q'_1, q'_2)) \mid \exists l_1, l_2,\ (q_1, l_1, q'_1) \in T_1 \wedge (q_2, l_2, q'_2) \in T_2 \wedge l = \Theta_{E_1 \cap E_2}(l_1, l_2)\}$,
- $I = I_1 \times I_2$ *et*
- $F = F_1 \times F_2$.

La synchronisation de deux automates A_1 et A_2 consiste à considérer l'évolution sur les deux automates en n'empruntant que des transitions synchrones. Remarquons qu'on considère également les transitions implicites $t_i = (q_i, \emptyset, q_i)$ pour ce calcul.

A.4 États

Historiquement, on considère les états comme un ensemble $Q = \{q_1, q_2, \ldots\}$. On peut cependant (à un renommage des états près) considérer qu'il existe un ensemble de variables booléennes (appelées *propriétés*) noté $\mathcal{P}$ tel que $Q \subseteq 2^{\mathcal{P}}$. C'est notamment le cas lorsque le modèle est représenté par des outils symboliques (voir [MR02]). Considérons deux automates A_1 et A_2 tels que leurs ensembles de propriétés sont disjoints : $\mathcal{P}_1 \cap \mathcal{P}_2 = \emptyset$. L'ensemble d'états de la synchonisation $A_1 \otimes A_2$ est $Q = Q_1 \times Q_2 \subseteq 2^{\mathcal{P}_1} \times 2^{\mathcal{P}_2} = 2^{\mathcal{P}_1 \cup \mathcal{P}_2}$. Un état $q \in Q$ est donc un ensemble de propriétés $q \subseteq \mathcal{P}_1 \cup \mathcal{P}_2$. L'ordre de la synchronisation n'a donc pas d'importance. Dès lors, on considère que $A_1 \otimes A_2 = A_2 \otimes A_1$. De même, on considère : $A_1 \otimes \cdots \otimes A_n = (A_1 \otimes \ldots) \otimes A_n = A_1 \otimes (\cdots \otimes A_n)$.

Ces propriétés sur les états sont particulièrements utiles lorsqu'on considère des synchronisations successives d'automates.

Annexe B

Format des fichiers de modélisation

Cette annexe donne le format des fichiers modélisant le comportement d'un composant. Le format est basé sur la norme XML. Nous donnons ici l'exemple de la pompe (étendu pour présenter tous les aspects).

Le fichier se trouve sous la balise `module` qui contient un champ `version` (actuellement 0.1) et un champ `name` qui correspond au nom du type de composant. Voir exemple B.1

```
<!DOCTYPE RageML>
<module version="0.1" name="pompe" >
```

Exemple B.1: En-tête

Le fichier se divise en trois parties : `ip`, `states`, `transitions`.

`ip` décrit les points de connexion du composant. On trouve quatre types de point : `ext`, `observations`, `output` et `input` (les deux derniers se trouvent sous les balises `outputs` et `inputs`). Chaque point dispose d'un champ `name` et de plusieurs balises `label` comportant le champ `name`. Sur l'exemple B.2, il y a quatre points de connexion. Le point de connexion `ext` est le point extérieur et peut recevoir les messages `fuite` ou `panne` modélisant l'apparition d'une fuite ou d'une panne.

La balise `states` décrit les états. Un état est représenté par son `label` et le champ `initial` est à 1 s'il est initial. Les autres champs sont des informations internes utilisées par *RAGeCreator*, le logiciel permettant de construire le modèle à l'aide d'une interface. Un état peut également avoir des points de connexion sains représentés par la balise `safe` et le champ `name`. On peut se référer à l'exemple B.3.

Les `transitions` sont chacune représentées par une `trans`. Une transition est générée par la réception d'un message (champ `label`) sur un point de connexion (champ `input`). Elle peut générer des messages (balise `outputs`). Chaque message (champ `label` de la balise `output`) est généré sur un point (champ `name`). La transition peut également générer une observation (champ `label` de la balise `observation`). Les autres informations servent la représentation interne de *RAGeCreator*. Voir l'exemple B.4 qui fournit également le pied de page.

```
<ip>
 <ext name="ext" >
  <label name="fuite" />
  <label name="panne" />
 </ext>
 <observations name="obs" >
  <label name="eleve" />
  <label name="faible" />
  <label name="zero" />
 </observations>
 <outputs>
  <output name="sortie" >
   <label name="eleve" />
   <label name="faible" />
   <label name="zero" />
  </output>
 </outputs>
 <inputs>
  <input name="interrupteur" >
   <label name="on" />
   <label name="off" />
  </input>
 </inputs>
</ip>
```

Exemple B.2: Informations sur les points de connexion

```
<states>
 <state width="50" sid="1" locX="451" locY="261" initial="0"
   label="ok,allume" />
 <state width="50" sid="3" locX="317" locY="472" initial="0"
   label="fuite,allume" />
 <state width="50" sid="2" locX="177" locY="475" initial="0"
   label="fuite,eteint" >
  <safe name="sortie" />
 </state>
 <state width="50" sid="4" locX="522" locY="477" initial="0"
   label="panne,allume" >
  <safe name="sortie" />
 </state>
 <state width="50" sid="5" locX="683" locY="479" initial="0"
   label="panne,eteint" >
  <safe name="sortie" />
 </state>
 <state width="50" sid="0" locX="455" locY="135" initial="0"
   label="ok,eteint" >
  <safe name="interrupteur" />
  <safe name="sortie" />
 </state>
</states>
```

Exemple B.3: Informations sur les états

```
 <transitions>
  <trans input="interrupteur" sid="0" tid="1" id="1" label="on" >
  <trans input="interrupteur" sid="1" tid="0" id="2" label="off" >
   <points>
    <point0 x="470" y="246" />
    <point1 x="526" y="200" />
    <point2 x="473" y="151" />
   </points>
   <outputs>
    <out name="sortie" label="zero" />
   </outputs>
   <observation label="zero" />
  </trans>

      [...]

 </transitions>
</module>
```

Exemple B.4: Informations sur les transitions

Table des figures

Liste des tableaux

Table des définitions

Bibliographie

[AFB+98] A. Aghasaryan, E. Fabre, A. Benveniste, R. Boubour et Cl. Jard. Fault detection and diagnosis in distributed systems: an approach by partially stochastic Petri nets. *Discrete Event Dynamic Systems*, 8(2):203–231, 1998.

[Alu99] R. Alur. Timed automata. Dans *Eleventh International Conference on Computer Aided Verification (CAV-98)*, pages 8–22, Trento, Italie, Juillet 1999.

[AU72] A. Aho et J. Ullman. *The Theory of Parsing, Translation, and Compiling*, volume 1, chapter Theory of translation, pages 223–233. Prentice Hall Professional Technical Reference, 1972.

[BBF+01a] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and Ph. Schnoebelen. *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer, 2001.

[BBF+01b] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and Ph. Schnoebelen. *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer, 2001.

[BBPP04] M. Bednarczyk, L. Bernardinello, W. Pawlowski et L. Pomello. Modelling mobility with petri hypernets. Dans *Seventeenth International Workshop on Algebraic Development Techniques (WADT-04)*, pages 28–44, Barcelone, Espagne, Mars 2004.

[BFHJ03] A. Benveniste, E. Fabre, S. Haar et Cl. Jard. Diagnosis of asynchronous discrete event systems, a net unfolding approach. *IEEE Transactions on Automatic Control*, 48(5):714–727, May 2003.

[BHFJ04] A. Benveniste, S. Haar, E. Fabre et Cl. Jard. Distributed monitoring of concurrent and asynchronous systems. *Journal of Discrete Event Systems*, 15(1), 2004.

[BHFJ05] A. Benveniste, S. Haar, E. Fabre et Cl. Jard. Fault diagnosis for distributed asynchronous dynamically reconfigured discrete event systems. Dans *Sixteenth International Federation of Automatic Control World Congress (IFAC-05)*, Prague, République Tchèque, Juillet 2005.

[BLPZ99] P. Baroni, G. Lamperti, P. Pogliano et M. Zanella. Diagnosis of large active systems. *Artificial Intelligence*, 110:135–183, 1999.

[BMS00] C. Barral, S. McIlraith et T.C. Son. Formulating diagnostic problem solving using an action language with narratives and sensing. Dans *Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR-00)*, pages 311–322, 2000.

[BY04] J. Bengtsson et W. Yi. Timed automata: Semantics, algorithms and tools. *Lecture Notes on Concurrency and Petri Nets*, (LNCS 3098), 2004.

[Cas93] C. Cassandras. *Discrete-event System, modeling and performance analysis.* Aksen Associates Publishers, 1993.

[CGLP03a] M.-O. Cordier, A. Grastien, Ch. Largouët et Y. Pencolé. Calcul de trajectoires utilisant les propriétés d'interversibilité. Dans *Sixièmes rencontres nationales des jeunes chercheurs en intelligence artificielle (RJCIA-03)*, pages 15–28, Laval, France, 2003.

[CGLP03b] M.-O. Cordier, A. Grastien, Ch. Largouët et Y. Pencolé. Efficient trajectories computing exploiting inversibility properties. Dans *Fourteenth International Workshop on Principles of Diagnosis (DX-03)*, pages 93–98, Washington, USA, 2003.

[CL99] C. Cassandras et S. Lafortune. *Introduction to Discrete Event Systems.* Kluwer Academic Publishers, 1999.

[CL01] M.-O. Cordier et Ch. Largouët. Using model-checking techniques for diagnosing discrete-event systems. Dans *Twelfth International Workshop on Principles of Diagnosis (DX-01)*, pages 39–46, Via Lattea, Italie, Mars 2001.

[CPR00] L. Console, Cl. Picardi et M. Ribaudo. Diagnosis and diagnosability analysis using PEPA. Dans *Fourteenth European Conference on Artificial Intelligence (ECAI-00)*, pages 131–135, Berlin, Allemagne, 2000.

[CT94] M.-O. Cordier et S. Thiébaux. Event-based diagnosis for evolutive systems. Dans *Fifth International Workshop on Principles of Diagnosis (DX-94)*, pages 64–69, 1994.

[CTJK97] M.-O. Cordier, S. Thiébaux, O. Jehl et J.-P. Krivine. Supply restoration in power distribution systems: a reference problem in diagnosis and reconfiguration. Dans *Eighth International Workshop on Principles of Diagnosis (DX-97)*, pages 27–34, Mont-Saint-Michel, France, 1997.

[DA94] R. David et H. Alla. Petri nets for modeling of dynamic systems – a survey. *Automatica*, 30(2):175–202, 1994.

[Dav] A. David. *UPPAAL2k: Small Tutorial. http://www.it.uu.se/research/group/darts/uppaal/tutorial.ps.gz.*

[DLT98] R. Debouk, S. Lafortune et D. Teneketzis. Coordinated decentralized protocols for failure diagnosis of discrete event systems. Dans *IEEE Conference on Decision and Control*, volume 4, pages 3763–3768, 1998.

[GCL04] A. Grastien, M.-O. Cordier et Ch. Largouët. Extending decentralized discrete-event modelling to diagnose reconfigurable systems. Dans *Fifteenth*

International Workshop on Principles of Diagnosis (DX-04), pages 75–80, Carcassonne, France, Juin 2004.

[GCL05a] A. Grastien, M.-O. Cordier et Ch. Largouët. Automata slicing for diagnosing discrete-event systems with partial ordered observations. Dans *Ninth Congress of the Italian Association for Artificial Intelligence (AI*IA-05)*, pages 270–281, Milano (Italy), Sep 2005. Springer-Verlag GmbH.

[GCL05b] A. Grastien, M.-O. Cordier et Ch. Largouët. First steps towards incremental diagnosis of discrete-event systems. Dans *Eighteenth Canadian Conference on Artificial Intelligence (AI-05)*, pages 170–181, Victoria (British-Columbia, Canada), Mai 2005. Springer-Verlag GmbH.

[GCL05c] A. Grastien, M.-O. Cordier et Ch. Largouët. Incremental diagnosis of discrete-event systems. Dans *Sixteenth International Workshop on Principles of Diagnosis (DX-05)*, pages 119–124, Pacific Grove, California, USA, Juin 2005.

[GH94] S. Gilmore et J. Hillston. The pepa workbench: A tool to support a process algebra-based approach to performance modelling. Dans *Seventh International Conference on Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS-94)*, pages 353–368, Vienna, 1994.

[Gue03] B. Guerraz. Construction de chroniques pour le diagnostic à partir d'une modélisation du système. Dans *Sixièmes rencontres nationales des jeunes chercheurs en intelligence artificielle (RJCIA-03)*, pages 125–137, Laval, France, 2003.

[HD05] J. Huang et A. Darwiche. On compiling system models for faster and more scalable diagnosis. Dans *Sixteenth International Workshop on Principles of Diagnosis (DX-05)*, pages 95–100, Pacific Grove, California, USA, Juin 2005.

[HG] Jane Hillston et Stephen Gilmore. Performance evaluation process algebra (PEPA). *http://www.dcs.ed.ac.uk/pepa/*.

[Hil94] J. Hillston. *A Compositional Approach to Performance Modelling*. Rapport de thèse, University of Edinburgh, 1994.

[JB05] G. Jiroveanu et R. Boël. Petri net model-based distributed diagnosis for large interacting systems. Dans *Sixteenth International Workshop on Principles of Diagnosis (DX-05)*, pages 25–30, Pacific Grove, California, USA, Juin 2005.

[KGM91] R. Kumar, V. Garg et S. Marcus. On controllability and normality of discrete event dynamical systems. *Systems and Control Letters*, 17:157–168, 1991.

[Lar00] Ch. Largouët. *Aide à l'interprétation d'une séquence d'images par la modélisation de l'évolution du système observé*. Rapport de thèse, Université de Rennes 1, 2000.

[LC00] Ch. Largouët et M.-O. Cordier. Combining observations and expectations: Application to the refinement of an image sequence. Dans *Fusion of Domain Knowledge with Data for Decision Support (Workshop UAI)*, Standford, CA, USA, Juin 2000.

[LZ03a] G. Lamperti et M. Zanella. Continuous diagnosis of discrete-event systems. Dans *Fourteenth International Workshop on Principles of Diagnosis (DX-03)*, pages 105–111, Washington, USA, 2003.

[LZ03b] G. Lamperti et M. Zanella. *Diagnosis of Active Systems*. Kluwer Academic Publishers, 2003.

[LZ04] G. Lamperti et M. Zanella. A bridged diagnostic method for the monitoring of polymorphic discrete-event systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 34(5):2222–2244, 2004.

[LZ05] G. Lamperti et M. Zanella. Monitoring and diagnosis of discrete-event systems with uncertain observations. Dans *Poster – Sixteenth International Workshop on Principles of Diagnosis (DX'05)*, Pacific Grove, California, USA, Juin 2005.

[Maz88] A. Mazurkiewicz. Basic notions of trace theory. Dans *School/Workshop on Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, pages 364–397, 1988.

[MBC+95] M. Marsan, G. Balbo, G. Conte, S. Donatelli et G. Franceschinis. *Modeling with Generalized Stochastic Petri Nets*. John Wiley & Sons, 1995.

[MR02] H. Marchand et L. Rozé. Diagnostic de pannes sur des systèmes à évènements discrets : une approche à base de modèles symboliques. Dans *13ème Congrès Francophone AFRIF-AFIA de Reconnaissance de Formes et d'Intelligence Artificielle (RFIA02)*, pages 191–200, Angers, France, Janvier 2002.

[Mur89] Tadao Murata. Petri nets: Properties, analysis and applications. Dans *Proceedings of the IEEE*, volume 77, pages 541–580, April 1989.

[NPW81] M. Nielsen, G. Plotkin et G. Winskel. Petri nets, event structures and domains, part i. *Theoretical Computer Science*, 13:85–108, 1981.

[PC05] Y. Pencolé et M.-O. Cordier. A formal framework for the decentralised diagnosis of large scale discrete event systems and its application to telecommunication networks. *Artificial Intelligence Journal*, 164(1-2):121–170, 2005.

[PCR01] Y. Pencolé, M.-O. Cordier et L. Rozé. Incremental decentralized diagnosis approach for the supervision of a telecommunication network. Dans *Twelfth International Workshop on Principles of Diagnosis (DX-01)*, pages 151–158, Via Lattea, Italie, Mars 2001.

[Pel93] D. Peled. On model checking using representatives. Dans *Fifth International Conference on Computer-Aided Verification (CAV-93)*, pages 409–423, 1993.

[Pen02] Y. Pencolé. *Diagnostic décentralisé de systèmes à événements discrets : application aux réseaux de télécommunications.* Rapport de thèse, Université de Rennes 1, 2002.

[RC02] L. Rozé et M.-O. Cordier. Diagnosing discrete-event systems : extending the "diagnoser approach" to deal with telecommunication networks. *Journal on Discrete-Event Dynamic Systems: Theory and Applications (JDEDS)*, 2002.

[Roz97] L. Rozé. *Supervision de réseaux de télécommunication : une approche à base de modèles.* Rapport de thèse, Université de Rennes 1, 1997.

[Sen98] R. Sengupta. Diagnosis and communication in distributed systems. Dans *Ninth International Workshop on Principles of Diagnosis (DX-98)*, pages 144–151, 1998.

[SM96] J. Sztipanovits et A. Misra. Diagnosis of discrete event systems using ordered binary decision diagrams. Dans *Seventh International Workshop on Principles of Diagnosis (DX-96)*, 1996.

[SPT04] A. Schumann, Y. Pencolé et S. Thiébaux. Diagnosis of discrete-event systems using binary decision diagrams. Dans *Fifteenth International Workshop on Principles of Diagnosis (DX-04)*, pages 197–202, Carcassonne, France, Juin 2004.

[SSL+95] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen et D. Teneketzis. Diagnosability of discrete event systems. *IEEE Transactions on Automatic Control*, pages 1555–1575, 1995.

[SSL+96] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen et D. Teneketzis. Failure diagnosis using discrete-event models. *Control Systems Technology*, 4(2):105–124, Mars 1996.

[SW00] R. Su et W. Wonham. Decentralized fault diagnosis for discrete-event systems. Dans *34th Annual Conference on Information Sciences and Systems (CISS-00)*, 2000.

[SW04] R. Su et W. Wonham. A model of component consistency in distributed diagnosis. Dans *Fifteenh International Workshop on Principles of Diagnosis (DX-04)*, pages 63–68, Carcassonne, France, Juin 2004.

[TCJK96] S. Thiébaux, M.-O. Cordier, O. Jehl et J.-P. Krivine. Supply restoration in power distribution systems — a case study in integrating model-based diagnosis and repair planning. Dans *Twelfth Conference on Uncertainty in Artificial Intelligence (UAI-96)*, pages 525–532, Portland, Oregon, USA, Août 1996.

[TLFi] TLFi. Le trésor de la langue française informatisé. développé par l'unité mixte de recherche ATILF (Analyse et Traitement Informatique de la Langue Française) du CNRS, *http://atilf.atilf.fr/tlf.htm.*

[TT03a] D. Thorsley et D. Tenekezis. Failure diagnosis of stochastic automata. Dans *Fourteenth International Workshop on Principles of Diagnosis (DX-03)*, Washington, USA, 2003.

[TT03b] G. Torta et P. Torasso. Automatic abstraction in component-based diagnosis driven by system observability. Dans *Fourteenth International Workshop on Principles of Diagnosis (DX-03)*, pages 394–400, Washington, USA, 2003.

[TT04] G. Torta et P. Torasso. The role of OBDDs in controlling the complexity of model based diagnosis. Dans *Fifteenth International Workshop on Principles of Diagnosis (DX-04)*, pages 9–14, Carcassonne, France, Juin 2004.

[Var95] M. Vardi. An automata-theoretic approach to linear temporal logic. Dans *Banff Higher Order Workshop*, pages 238–266, 1995.

[WYL04] Y. Wang, T.-S. Yoo et S. Lafortune. New results on decentralized diagnosis of discrete event systems. Dans *42nd Annual Allerton Conference on Communication, Control, and Computing*, Illinois, USA, 2004.

[Yov97] S. Yovine. Kronos: A verification tool for real-time systems. *International Journal of Software Tools for Technology Transfer*, 1(1/2):123–133, 1997.

Résumé

Le cadre de cette thèse est la surveillance et le diagnostic de systèmes dynamiques reconfigurables modélisés comme des systèmes à événements discrets. Ces systèmes sont constitués d'un ensemble de composants interagissants. Une reconfiguration correspond à une modification (ajout, suppresion, remplacement) des composants du système et des connexions entre les composants.

Le système fournit des observations directement ou par l'intermédiaire de capteurs qui le surveillent. Il est possible d'interpréter ces observations pour inférer l'état du système et les dysfonctionnements de ses composants. Nous utilisons une approche basée sur le modèle du système. Le modèle est un automate dont chaque trajectoire représente un comportement possible du système. La tâche du diagnostic se définit alors comme trouver l'ensemble des trajectoires du système qui expliquent les observations. Les observations reçues étant différentes des observations émises, nous représentons les observations émises par un automate dont toute trajectoire correspond à une séquence possible d'émission étant données les observations reçues. Le diagnostic se définit alors comme la synchronisation entre le modèle et l'automate des observations.

Pour permettre un diagnostic par périodes, nous avons défini une structure de données appelée *chaîne d'automates* qui permet de représenter un automate *par morceaux*, en particulier l'automate des observations. Il est possible d'effectuer un raisonnement sur chaque morceau de l'automate et de reconstruire ensuite le diagnostic de la période complète. Cette vision modulaire permet un diagnostic incrémental et en-ligne, consistant à construire le diagnostic pendant que le système délivre ses observations.

Nous avons utilisé ces résultats pour permettre le diagnostic de systèmes reconfigurables. Nous avons d'abord défini la reconfiguration et ses conséquences sur le système. Ensuite, nous avons utilisé les chaînes d'automates et découpe les observations pour obtenir des morceaux correspondant à chaque configuration.

Nous avons validé ces résultats par une plateforme de diagnostic décentralisé.

Mots-clé

Diagnostic à base de modèles, systèmes à événements discrets, intelligence artificielle, systèmes reconfigurables, diagnostic en-ligne